tanulmányok

100/1979

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

OPERÁCIÓS RENDSZEREK ELMÉLETE
V. VISEGRÁDI TÉLI ISKOLA

Tanulmányok 100/1979.

Felelős kiadó:

DR VÁMOS TIBOR

MTA Számitástechnikai és Automatizálási Kutató Intézete
MTA Számitástudományi Bizottsága

ISBN 963 311 097 1
ISSN 0324-2951

79-1835 Tempó Sokszorositó - 300 pld.
F.v.: Vojvoda György

A konferenciát a "Számitástechnika tudományos kérdései" c. többoldalu akadémia együttmüködés keretében rendeztük.

Конференция была проведена в рамках многостороннего сотрудничества академий социалистических стран по проблеме "Научные вопросы вычислительной техники"

Conference was held in the frame of the multilateral cooperation of the academics of sciences of the socialist countries on Computer Sciences.

TARTALOMJEGYZÉK

INTERFERENCE PROBLEM IN DISTRIBUTED COMPUTING

Jose Busta,* Serge Miranda**

INTRODUCTION

In local DBMS one of the major objectives at the implementation level was to avoid data physical redundancy.

This entailed the requirement to share data among users having different needs and therefore to control update concurrency which we call "external integrity" ; it is important to note that every proposed solution, included in the Univac's DBTG-like DBMS (DMS 1 100) where the initial "monitoring mechanism" was given up, is based on a LOCKING MECHANISM (low or high level). From locking, stems the potentiality of deadlock (locking is one of the five necessary conditions for deadlock). The major purpose of this locking mechanism is to maintain data base consistency.

In distributed data bases, physical redundancy among remote parts of the scattered data was re-introduced, mainly for availability and reliability reasons. This redundancy can be "identity", "extension" or "restriction" of a given entity. Mutual consistency of these "duplicated" entities must be maintained. This will be performed via an external integrity mechanism which will control the concurrent access to the data.

We address in (1) the generalization of this issue to "partitionned data bases" where different data modules are located throughout the network (without being necessarily semantically tied).

We defined a global locking mechanism to tackle the external integrity problem.

External integrity is only one aspect of the interference problem ; the second is concerned with UPDATE PROPAGATION through the different functional levels (external, conceptual, access-path, encoding, physical device) of a DBMS (local or distributed).

* Computing Center University of Santiago de Compostela, Spain

** Ceriss-University of Toulouse I, France

This problem was only partially solved :

- SYSTEM-R defined simple rules between the external and conceptual levels like the "rectangular and "uniqueness" one ;
- INGRES defined simple mapping algorithms of the same type ;
- IBM-Los Angeles system included "reflection rules "within the definition of external entities.

Only PAOLINI in the university of MILANO tried a uniform and rigorous approach based on abstract data types to deal with this problem.

In the remainder of this paper we will present our solution and modelization to the external integrity issues for distributed duplicated entities. We will focus our attention on a fail-safe computer network.

MAJOR FEATURES OF OUR ALGORITHM

The significant characteristics of our synchronization protocol are :

- strong consistency ;

at a given point of time, every open copy is in the same consistent state (even in case of any failure in the distributed system).

- global locking ;

our protocol implements mutual exclusion this requires the definition of a coordination protocol among remote controllers. We choose a preclaiming strategy (which appears to be the only prevention solution possible in a distributed environment) to deal with deadlock ; as a consequence our protocol will encompass two steps of synchronization.

General purpose network oriented ;

its control is decentralized. There is no privileged controller a priori ; there is a master-site determination during the first step based on a priority system (including parameters like TIME-STAMP, REJECTION number, less-priority transaction list,...). Every transaction will be processed after an indefinite but finite time. Our protocol must be efficient (transmission overhead in the order of 5 (n-1) messages with n, number of copies, but with a response time independant of n due to parallel transmission/processing) and robust. This requires that every global operation be two-PHASE. At last our protocol is speed independent by the integration of acknowledgement messages.

APPEAL OF DISTRIBUTED CONTROL

Centralized control is attractive due to the general simplicity of the corresponding algorithm (minimal message transmission, simplification of the conflict resolution, possibility to use deadlock detection algorithms,...) and turns to be adequate for star-like network structures or centralized applications.

However such a solution is not inherently reliable in case of central site failure or isolation and does not meet the objectives of a general purpose computer network (i.e ARPANET, CYCLADES, EIN,...) like modular growth, crash confinement, reliability, local autonomy and load sharing.

For these above requirements to be true it is necessary to provide all the processes with equal power. Therefore we discarded the centralized control approach and define a protocol where all controllers are initially equal. This distributed control makes use of a priority system which will be used for deadlock prevention and master site determination.

A protocol using a distributed control can be made extremely robust. The final advantage of the distributed approach is that the proof of correctness of such an algorithm (identical in each controller) is easier to conduct.

Our protocol is "message- driven" ; the activity of each autonomous controller will be coordinated by synchronization messages and a sequence of actions will occur as a consequence of the reception of a dedicated message or a group of messages.

There are two steps of synchronization corresponding to :

- the election of a "master controller" and the resolution of conflicts which can occur among concurrent transactions.

- the global locking-update-unlocking is then performed by the master controller with the other controllers called "slaves".

STATE DIAGRAM

An algorithm is best staded via some abstract means which is accurate and implementation independent .

Each message causes a transition among states of locks (global and local). This is a reason why we use an automata-like formalism to represent our algorithm.

We have 3 different resource states (or lock states) :

- free (F) : there is no global synchronization process in progress. Any update request can be initiated.

- occupied (O) : there is a global synchronization process in progress but the master controller, elected during the first step of synchronization can still switch. This state corresponds to the protected mode and an update request can still be considered.

- blocked up (B) : there is a global synchronization process in progress and the resource (global or local) is blocked up for a given master controller. This state corresponds to an exclusive mode.

The automata representing the algorithm, in a fail safe network is the 5-uple $\langle X, Q, F, qo, S \rangle$ where

$X = \{\Lambda, DBD, RDBD, DMS, RDMS, FDB, OINT, UPR, RUPR\}$

$Q = \{Fi, Oi, Oij, Bi, Bij\}$

$F = qo = \{Fi\}$

S : transition function depicted in figure 1

Notation of figure 1

Xij : lock state with $X \in \{O,B\}$ means that the controller i is in the state X for the controller j

$\longrightarrow$: transition arrow between lock states

- labelled with transition-id
- numbered with the order it appears in the protocol

transition-id = [received message (s) (significant parameter), transmitted message(s)]

α.Aij : α A-type message (s) from controller i to controller j (received or transmitted)

A can be of the following type :

UPR : update request sent by the external user and its acknowledgment RUPR

DBD : demand for distant blocking and its ACK , RDBD

DMS : demand for synchronized update and its ACK, RDMS

OINT : order of interrupt

C : message counter $\in [1, n-1[$

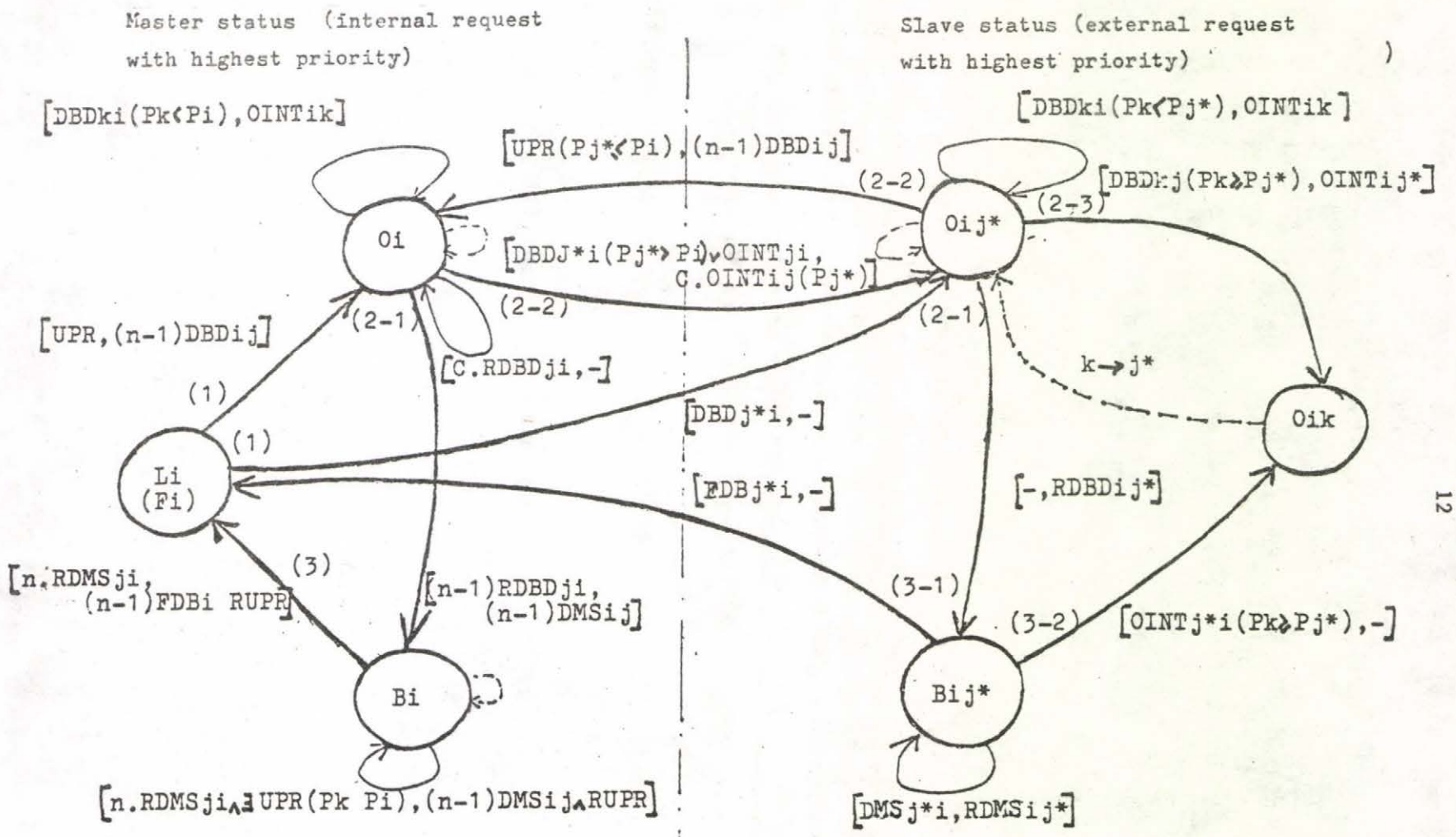


Figure 1 State diagram of the i th controller
(fail-safe network)

USE OF ABSTRACT DATA TYPES TO FORMALIZE THE ALGORITHM

Locking in a distributed data base cannot be reduced only to :

- a data structure (like the "rectangular view" defined in System-R...) or

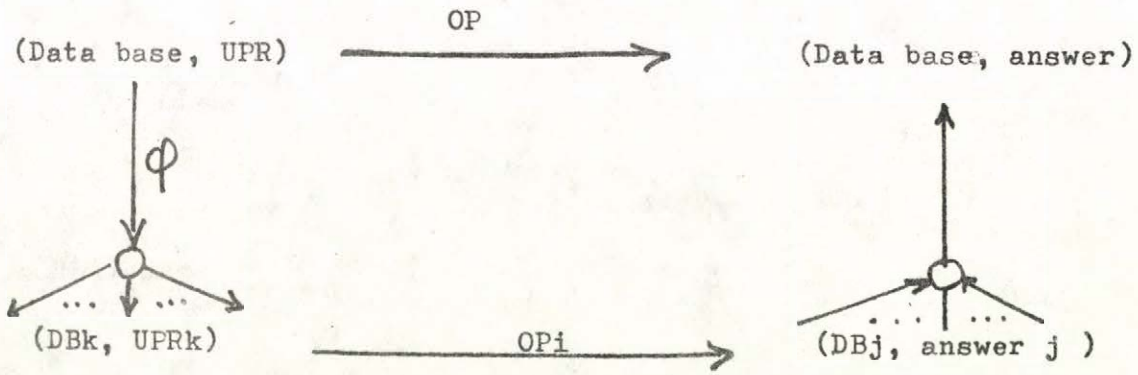
- a coordination protocol (like in duplicated file systems...)

which would lead to a sub-specification. Hence the appeal of abstract data types (ADT) which allow to consider locking both as a data structure and as a set of operators defined on it.

Since the data structures involved in locking a DB are complex we will choose the algebraic approach (instead of the propositional approach more suitable to simple structures) to describe the different ADTs attached to each functional level of a distributed DBMS (at least, five levels have been identified (1)).

Therefore, the synchronization system will be seen as a set of ADTs depicted as homomorphic Σ -algebras.

We will have a twofold mapping to define for the operators and the data-structures



In the particular case we consider (duplicated file system), locking is reduced to the second type of locking, taken into account in the coordination protocol between the global level and the local level (concerning each local copy).

We depict in figure 2 and 3 both ADTs atta ched to the previously defined protocol. We use the syntax of OBJ which is an object oriented language defined in UCLA (6).

We use also the fact that a finite state automata (fig. 1) represents a particular class of Σ -algebra and we will consider the lock states (global and local levels) as the carriers of the corresponding algebra.

To each of these levels, we associate two ADTs, SYNCG and SYNCL which are Σ -algebras defined by the 5-uple

$\langle S, \Sigma, \Xi, \mathcal{E}, K \rangle$ where

- S is the set of sorts
- Σ is the set of operators defined on the sorts
- Ξ is the set of error operators
- $\mathcal{E}$ is the set of equations of the Σ operators
- K is the set of equations of the Ξ operators

Notation for the figures 2 and 3

[OP1, OP2] indicates a running parallelism between the operators OP1 and OP2

α. OPi expresses that the operator i will be executed in parallel on α remote controllers

ATT(UPR(PK)) there is scheduler call and the request with parameter PK is delayed. This action is external to the protocol

MAJL : update operator of the local file (external to the protocol).

OBJECT SYNCG

SORTS E,D

OPS

ODBD : E,D $\longrightarrow$ E, E $\vee$ E'
ORDBD: E,D $\longrightarrow$ E
ODMS : E $\longrightarrow$ E, E'
ORDMS: E,D $\longrightarrow$ E,E'
OFDBD: E $\longrightarrow$ E
OINT : E $\longrightarrow$ E
OUPR : E,D $\longrightarrow$ E,E $\vee$ E'
ID : E $\longrightarrow$ E

ERROR-OPS

ERR1 : $\longrightarrow$ E,D
ERR2 : $\longrightarrow$ E,D
ERR3 : $\longrightarrow$ E,D
ERR4 : $\longrightarrow$ E
ERR5 : $\longrightarrow$ E
ERR6 : $\longrightarrow$ E,D
ERR7 : $\longrightarrow$ E,D
ERR8 : $\longrightarrow$ E
ERR9 : $\longrightarrow$ E
ERR10: $\longrightarrow$ E
ERR11: $\longrightarrow$ E

VARS

Oi : E , Bi : E , Li : E
C : D , Pi : D , - : D

$\bigwedge$ = Oi v Bi v Li avec i $\in$ [1,n]
C $\in$ [1,n-1] , Pi $\in$ [1,n]

SPECS

OUPR (Li,Pk) = [ODBD (Li,-), (n-1) ODBD' (Λ',Pk)]
OUPR (Oi,Pk) = IF Pk ⩽ Pi THEN [ATT (UPR (Pk)),ID(Oi)]
ELSE [ATT (UPR (Pi)),ID(Ok)] (1)
OUPR (Bi,Pk) = [ATT (UPR (Pk)), ID (Bi)]
ODBD (Li,-) =(Oi,Λ)
ODBD (Oi,Pk) = IF Pk ⩽ Pi THEN [ID (Oi), OINT (Ok)]
ELSE [ID'(Oik,Pk), ORDBD (Ok,-)
+ C x OINT' (Bji,Pk)]
ORDBD (Oi, c)= IF C = n-1 THEN ODMS (Bi) ELSE ID (Oi)
ODMS (Bi) = [ID (Bi) + MAJL , (n-1) ODMS' (Bji)]
ORDMS (Bi,c) = IF C = n-1 THEN [OFDBD (Bi), (n-1) OFDBD'(Bji)
+ ORUPR]
ELSE ID (Bi)

OFDBD (Bi) = Li
OINT (Oi) = OivOij
ID (Λ) = Λ

ERROR-SPECS

ERR1 = ODBD (Bi,-)
ERR2 = ORDBD (Li,-)
ERR3 = ORDBD (Bi,-)
ERR4 = ODMS (Li)
ERR5 = ODMS (Oi)
ERR6 = ORDMS (Li,-)
ERR7 = ORDMS (Oi,-)
ERR8 = OFDBD (Li)
ERR9 = OFDBD (Oi)
ERR10 = OINT (Li)
ERR11 = OINT (Bi)

TCE JBO Figure 2 Abstract data type, SYNCG (global level) written in OBJ

(1) The local controller waits for (n-1) RDBD ; the master switch, located on the same site, is transparent to the remote controllers.

OBJECT SYNCL

SORTS E',D'

OPS

ODBD' : E',D' ⟶ E',E
ORDBD' : E' ⟶ E'
ODMS' : E' ⟶ E'
OFDBD' : E' ⟶ E'
ORDMS' : E' ⟶ E'
OINT' : E',D' ⟶ E'
OUPR : E',D' ⟶ E',E
ID' : E' ⟶ E'

'ERROR-OPS

ERR1 : ⟶ E',D'
ERR2 : ⟶ E'
ERR3 : ⟶ E'
ERR4 : ⟶ E'
ERR5 : ⟶ E'
ERR6 : ⟶ E'
ERR7 : ⟶ E'
ERR8 : ⟶ E',D'
ERR9 : ⟶ E',D'

VARS

Oij : E' , Bij : E' , Lj : E' $\quad \Lambda' = Oij, Lj$
\- : D' , Pi : D'

SPECS

ODBD' (Lj,Pi) = [ID'(Lj,Pi),ORDBD (Oi,-)]
ODBD' (Oji,Pk)= IF Pk ≤ Pi THEN [ID'(Oji,Pi),OINT(Ok)]
ELSE [ID'(Oji,Pk),OINT (Oi)÷ ORDBD(Ok,-
ORDBD' (Oji) = Bji
ODMS' (Bji) = [ID'(Bji,-),MAJL]
ORDMS' (Bji) = [ID'(Bji,-),ORDMS (Bi,-)]
OFDBD' (Bji) = Lj
OINT' (Oji,-)= Ojk
OUPR' (Oji,Pj)= IF Pj ≤ Pi THEN [ID'(Oji,Pi),ATT(UPR(Pi)
ELSE [ID(Oj),(n-1)ODBD' (Λ',Pk)]
OUPR' (Bji,-) = [ATT(UPR (-)),ID' (Bji,-)]
ID' (Oji,Pk)= Ojk
ID' (Lj,Pi) = Oji
ID' (Bji,-) = Bji

ERROR-SPECS

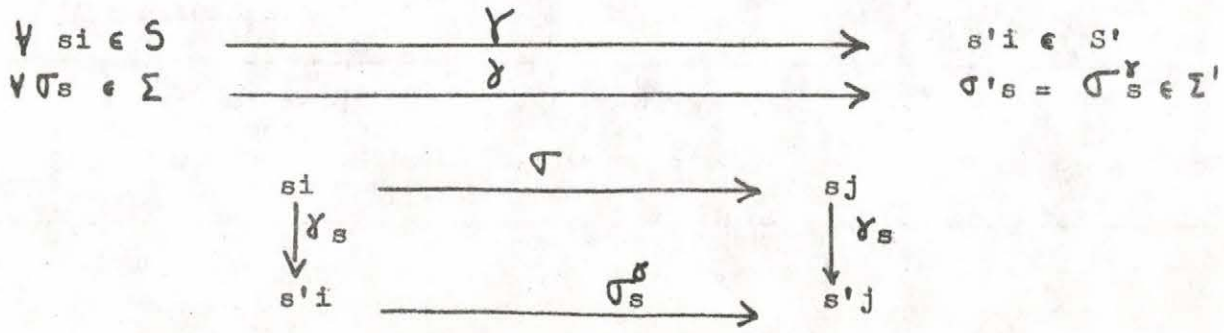
ERR1 = ODBD' (Bji,-)
ERR2 = ORDBD'(Lj)
ERR3 = ORDBD'(Bji)
ERR4 = ODMS' (Lj)
ERR5 = ODMS' (Oji)
ERR6 = OFDBD'(Lj)
ERR7 = OFDBD'(Oji)
ERR8 = OINT' (Lj,-)
ERR9 = OUPR' (Lj,-)

TCEJBO

Figure 3 Abstract data type, SYNCL (local level) written in OBJ

We can easily show that SYNCG and SYNCL are Σ-homomorphic with the natural morphism such as Li $\longrightarrow$ Lj , Bi $\longrightarrow$ Bji, Oi $\longrightarrow$ Oji $\vee$ Bji (the latter corresponds to the fact that the local resource can be in the blocked state while the global resource is still in the occupied state (waiting for all acknowledgements)).

As a matter of fact each of the following diagrams commutes :



The demonstration is straight forward.
If we choose the Σ-algebra SYNCL as the "reference algebra" (homomorphic to the implementation algebra) therefore the global -algebra SYNCG will be implementation correct.
In order to be rigorous, we will have to show that SYNCL is an "initial algebra" (6) (specification correct) and that there exists an isomorphism between SYNCL and SYNCG in order to make SYNCG "initial".

PERFORMANCE COMPARISON

Response time	Transmission Overhead	Algorithms
2 N. NT + T_{α} + (2NT.N)	2N + Nα + (2N.)	ELLIS's (5) "Ring structured solution"
NT (4 + $\frac{N}{2}$) + T_{α} + (2NT) (best case)	$\frac{3N}{2}$ + 2 + Nα + (2N) (best case)	THOMAS's (4) "Algorithm without global locking"
5 NT	3 N- 1	MENASCE's (3) "centralized control algorithm"
5 NT + T_{α} (2 steps)	5N - 3 + Nα(2steps)	OURS

NT : Network Time

T_{α} : overhead due to unsuccessful attempts

N : Number of cooperating controllers

( ): expression due to robustness integration

Nα: overhead due to previous rejections

Our protocol compares very well with a centralized solution both in terms of response time and transmission overhead. When we consider Ellis's and Thomas"s solutions we can notice that their transmission overhead is low ; it is not at all the case with their response time which depends on N and which grows linearly with it ; this is due to serial transmission which is avoided in our protocol.

CONCLUSION

We used automata and abstract data types to formalize the global locking in a distributed data system (fail-safe) ; such a formalization enabled us to prove the correctness of our algorithm, (part of it) . We prove that our algorithm is deadlock free by using a direct formalism (1).

We will use the abstract data type concept to incorporate the cases where the underlying network is unsafe (2).

The solution based on abstract data types will also be developped to tackle distributed data base issues (entity description attached to each functional level of a DBMS, basic functions,...) in a uniform and rigorous fashion.

References

(1) POPEK Gerry, MIRANDA Serge

"A decentralized robust algorithm for update management in a distributed system of computers" CERISS and UCLA research report, December 1978

(2) MIRANDA Serge

"Use of automata and abstract data types to formalize a coordination protocol in a distributed duplicated file system (unsafe net,,ork case)". CERISS research report, (forthcoming)

(3) MENASCE D., POPEK G. , MUNTZ R.

"A locking protocol for resource coordination in distributed systems" UCLA research report, 1977

(4) THOMAS R.H.

"A solution to the update problem for multiple copy data bases which use distributed control"
BBN report, 3340, July 1975

(5) ELLIS, C.A.

"A robust algorithm for updating duplicated data bases"
1977 workshop on distributed data management systems, Berkeley, pp 146-160, 1977

(6) GOGUEN J.A. and AL

"An initial algebra approach to the specification, correctness and implementation of abstract data types".UCLA research report, 1976, (To appear in current trends in programming methodology, Vol 3, Data structuring, Prentice Hall)

COMPUTER NETWORK EVOLUATION BY PROGRAM SIMULATION

Y.Bashin, I.Gorelik, H.Pranevitshius
USSR

0. ABSTRACT

The paper discusses some points of effective inspection of host, consisting of program compatible computers of various performances. It is studied by simulation the host structures, where job allocation among computers is carried out by a communication processor, the most productive computer, and a minicomputer as a foreman.

1. INTRODUCTION

Prompt service offers the greatest promise for network users and naturally the network efficiency there-with appears to be the final result of man-machine dialogue, i. e. user "thinking rate" and machine performance.

Prompt service allows each user to debug programs, get information and perform calculations as quickly, as if all resources were at its own disposal.

But for this service the network is to guarantee quick running of jobs, which is fairly hard in the real situation, where due to a factor of undecipherable ("bursty") arrivals, equipment failure etc. can happen some unusual overloads of the system elements, namely of communication channels, computers and so on.

Let us identify the following definitions [1].

Definition 1. Response time is a difference between access (of the network) of the job last character and its first character readout.

Definition 2. Peak load of the host is a quantity of grouped arrivals, whose number and parameters are conditioned by exceeding the accepted response time, if only for a

single arrival.

If the performance of a communication channel to transfer information to the host and that of the host are related as follows:

$$K < \frac{aC\left[N + 1 + \sqrt{N^2 + C^2\sigma^2 + 1}\right.}{2N - C^2\sigma^2} \quad , \qquad (1)$$

where K is a data processing rate (oper.per.sec.) and C being the information rate in bytes per sec. (Poisson arrivals with the rate of C), N denotes maximal arrivals, available in an accepted quantity. a is a mean labour--consumption per a job (operations per an information byte) and σ being the mean square variance of job labour--consumption, then in this case overload is impossible in the host.

In order to eliminate the overload one must improve host performance, which can be attained by the following menas:

- supplement an on-line computer to the system
- conversion to the special-purpose computers and finally
- single large computer systems (computer complexation).

Taking m be the equally accessed on-line computers of the host, which have random, independent arrivals, we shall see the next overload probability of the j-th computer (P_j), providing again independent access of various users and even computer capabilities, as follows

$$P_j = C_L^{l_j} p^{l_j} (1 - p)^{L-l_j} \quad , \qquad (2)$$

where p is an access probability at the time instant t and l_j being the sum of jobs, causing a peak load in the j-th computer. L denotes the sum of jobs, available on the host

by the instant t. Thus probability of peak loads in the host is equal

$$P_n = 1 - \prod_{j=1}^{m} \left[1 - C_L^{l_j} p^{l_j} (1 - p)^{L-l_j}\right]. \quad (3)$$

Analysis of the 3rd equation points to the fact, that a substantial degradation in probability of peak loads can be only attained by a great number of host computers, that is we see, that a correction of the system response time by simple admission of new computers is rather costly.

Network operation has shown, that it is impractical to use such large systems as main computers are, for data transmission with its functions of controlling data flows; that it is more advantageous to use for that function the special-purpose and cheaper mini-computers, namely communication processors, which is confirmed by the results of calculation of computer time. They have shown that it takes 15 to 30% of the computer time to do sums of data transmission and this value may be visible derated (to 1-4%) by introducing the communication processor into the system. We also consider the gain in reduction in the internal storage of the main computer, occupied by communications so far, and asset in that the processor can control the rate of arrivals besides, for example to decrease the rate of terminal operation, thus somewhat correcting the response time [2].

We kept, that most effective are special-purpose communication processors, allocating jobs among computers (of the host) and as a result of this approach we give the equation of peak load probability P_n, equal in this case 3

$$P_n = \frac{P_o \rho^l m^m}{m!} \quad , \quad P_o = \left[\frac{(m\rho)^m}{(1-\rho)m!} + \sum_{L=0}^{m-1} \frac{(m\rho)^L}{L!} \right]^{-1} \qquad (4)$$

where ρ is a traffic rate.

The equation (4) is true, if only the following assumptions behind this are counted for:

- computer equal performance
- Poisson arrivals and
- exponential service time.

If they are disrupted, for example, there are uneuqal performances then (4) becomes more complicated, that is, one must rewrite it in the next form for P_n, provided we have a host with two computers with corresponding performances μ_1, μ_2, i.e.

$$P_n = \frac{\psi^n}{1 + 2\psi} \cdot \frac{1 + L}{L} \left[1 + (1 + L)\psi - (1 - L)\phi \right] P_o \; ; \qquad (5)$$

$$\psi = \frac{\lambda}{\mu_1 + \mu_2} \; ; \qquad L = \frac{\mu_2}{\mu_1} < 1 \; ;$$

$$P_o = \frac{1 - \psi}{1 + \frac{\psi}{1 + 2\psi} \cdot \frac{1}{L} \left[1 + (1 + L^2)\psi - (1 - L^2)\phi \right]}$$

where ϕ is a probability of a single arrival at the first computer and $1-\phi$ being the probability of a single arrival at the second computer. λ denotes the arrival rate. The equations (4) and (5) come from the queueing theory, but it should be observed that for more comprehensive survey of network performance one must take into account different rates of data input and output, affecting on the processor, different job priorities, the impact of computer operating system etc.

Thus purely proceeding from the formulas of queueing theory, i.e. (4) and (5), we are not able to adequately predict in terms of host computer integration the whole

system operational factors, however, we still are able to conclude that computer integration, together with dynamic allocation of jobs are mostly promising in network evolution.

It is the aim of this work to examine the performance of some perspective host projects by simulation.

2. CONCEPTUAL MODELS OF THE HOST

The host includes m heterogeneous computers, varying in speed, specialization, capacities of internal storage etc, but each computer has a general field of memory of module organization, one of the modules being for storing information concerning computer condition. By a condition is meant the length of the input and output queue; the system also incorporates a communication processor, providing host and man-machine interface, reduction of incoming data and message rearrangement for transmission into network. To this must be added the special-purpose functions of the processor, namely interpretation of message priority and their routing, assignment of the internal storage zones for incoming messages. Besides it also provides in the system under survey the rate of information transmission at about 48000 baud, and at last, then it is done by one of the main computers of the host or mini-computer (the foreman).

Model 1. In this case the communication is thus to operate as a foreman and be then connected to the internal storage of all computers (Fig.1). On completion of the assignment of the needed computer, the latter is notified of it by processor via internal storage; thus transferred information consists of the list of modul numbers, which carry data about jobs for the given computer.

In case of data output (to the network) the computer communicates it to the processor via the internal storage again, by it indicating the list of external modules, where

the output information is stored.

Information about the current computer condition is automatically updated completion of processing a sequential problem by each computer.

Model 2. The foreman service is now performed by one of the main computers and the processor is in this time only connected to the foreman internal storage (Fig.2), transmitting jobs of a higher priority to foreman (the latter is also to handle this kind of jobs); jobs of a lawer priority are to be allocated by it among other host computers for execution, based on their current conditions. Thus rearranged jobs (requests) are distributed by the processor into the special module of the external storage, whereupon it signals the foreman the end of procedure, after which by screening the information about current conditions of other computers, the foreman finally arranges queues of requests before each computer, informing by it the computer about queue occurrence.

Model 3. The foreman service is now carried out by the special-purpose computer, purged of job processing, the internal storage of the processor is connected by it to the memory of all computers (Fig.3). The foreman is then also provided with the external storage access, storing data about the current condition of all computers; on completion of rearrangements the channel-based request before processing, it is placed in the general-purpose module by the processor, which signals the foreman about the end; the latter by interpreting the situ and queus rearrangement in priorities, then also signals the processor the need of data removal and reports by it the list of modules for the input--aimed external storage. After the end of input operation, the processor signals it and then the foreman carries out job distribution, based on the computer current condition, the results of which are delivered to each computer, again via internal storage.

Information output is realised by computer, which signals the foreman the need of data removal;, the later by reviewing free zones of the external storage informs (via internal storage) about the addressed of output modules.

3. HOST PROGRAM MODEL

The system under review is rather complicated and so the implementation of a program model, simulating the system operation as a whole is essentially a difficult task for realization, therefore we suggest, the whole system be presented as a set of interacting subsystems, each of which is a piecewise linear aggregate [5,6], with some number of input/output facilities, interfaced by a given way and described individually by a method of control sequences [7], allowing to declare the aggregate mathematically, that is in turn easily program realisable in computer.

This approach has given good results and some of its gains (as compared to other simulations) are as follows:

- a means for practical realization of more complicated program models;
- the system description in mathematical terms, that makes it easier preparing the program of simulation;
- a means for simultaneous work of several programmers, who are able now to handle models of single subsystems;
- there is no need in specialized algorithmic languages for simulation, as it is quite enough to know common program languages of higher level, for example FORTRAN and PL1.

3.1. Fig.4 shows the aggregate system of host, which includes the following:

- A_1 is a job generator (JG), which rearranges a stream of data, arriving from the network into a host and also

generates the following operational factors of jobs as priority, amount of input/output information, calculations etc.

- A_2 is a special-purpose communication processor, later called software/hardware interface (SHI), which has the next main parameters, as speed, internal storage capacity, number and speed of input/output channels;

- A_3 is a job dictributor (JD), which in the case of the 1st and/or 2nd conceptual model generates the output signal "REQUEST", based on which JD will respectively demand resources of the SHI or one of the computers; in the case of the 3rd conceptual model the JD requests for its own resources. The JD parameters are an amount of computations for solving allocation problems, the speed of its processor and also an algorithm to performs job allocation;

- A_4 is a general-purpose module of the external storage (GMES), which stores requests for host computers and the execution results, also makes exchange of information between itself and computers or/and SHI via common input/output channel;

- A_{4+i} is the i-th computer of host, where $i = \overline{1,m}$. All computers are multiprogrammed and with the operation system (OS/360); finally the speed of the processor is assumed to be the main computer parameter.

3.2. Each aggregate is a system of N input and M output poles the process of functioning of which is defined by a multitude of time instances, that is $T = \{t_o, t_1, t_2, \ldots, t_m, \ldots\}$, i.e. the instants, when one of the node events occurs. A multitude of node events is then subdivided into the two non-overlapping subsets, namely E' and E'', $E' \cup E'' = E$, $E' \cap E'' = \emptyset$, where the subset $E' = \{e'_1, e'_2, \ldots, e'_N\}$ includes events, comprising the arrival of an input signal, respectively $X_1, X_2, \ldots, X_N$; the subset $E'' = \{e''_1, e''_2, \ldots, e''_f\}$ includes events, which occur as a result of the aggregate

autonomy; and at last events e_i are supposed to be the completion of the i-th process, taking place in aggregate. The multitude T is subdivided respectively into the subset of T' moments of input signal arrivals and the subset of T'' instants, when there follow events as follows $e_i'' \in E$, $T' \cup T'' = T$.

The aggregate internal state is described by vector $\Theta(t_m)$, varying in time, that is

$$\Theta(t_m) = \{ \chi_1(t_m), \chi_2(t_m), \ldots, \chi_p(t_m) \} \qquad (6)$$

We assume that it is given a controlling sequence of values (not negative) for the aggregate $\{ T_{i,j}, j = \overline{1,f} \}_{j=1}^{\infty}$, which fully determines the process of the system operation.

For all events e_i, $i = \overline{1,f}$, there are constructed controlling $S_i(t_m)$ and checksums $V_i(t_m)$, having the next meaning

$S_i(t_m)$ = at the starting moment of the i-th process (in the aggregate), if at the instant of t_m the latter is in progress ∞, otherwise.

$$V_i(t_m) = \begin{cases} S_i(t_m) + T_{i,j}, & \text{if} \quad S_i(t_m) \neq \infty \\ \infty \ , \text{otherwise} \end{cases}$$

With a set of checksums $V_i(t_m)$, $i = \overline{1,f}$, we can define the instant, when there occurs a sequential event from the subset E'', i.e.

$$t''_{m+1} = \min_{1 \leqslant i \leqslant f} \{ V_i(t_m) \} \qquad (7)$$

The type of the event occured and transfer-of-control statement (of the aggregate new state) are determined by a

number of checksums, but since the incoming signals also change its state, then in general the aggregate transfer into another state can be expressed as

$$\Theta(t_{m+1}) = U_i \left[\Theta(t_m), e_i\right], \qquad e_i \in E \tag{8}$$

Output signals ϕ_i, i = 1,M, can be only arranged at the instants t_m of node event occurances and are dependent form the internal system state $\Theta(t_m)$ and input signals X_n, that is

$$\phi_i = G_i \left[\Theta(t_m), e_j, X_n\right] \tag{9}$$

$$1 \leqslant i \leqslant M, \qquad 1 \leqslant n \leqslant N, \qquad e_j \in E.$$

Consider the aggregation system of K-aggregates (including those of external medium), where definite limits are imposed on interaction of system elements, most important of which are the following [5]:

- transferring signals among the aggregates is instantly carried out;
- signals have zero length and keep in themselves some information;
- the aggregate cannot accept input signals, coming simultaneously in the same poles from various sources.

Let the k-th aggregate have N_k input and M_k output poles and the total number of links to transfer signals among the, shall be denoted by L.

Let the matrix be $R = \|r_{i,j}\|$, $i = \overline{1,L}$, $j = 1,2$ of input poles in the sense that its elements have the following meaning:

$r_{i,1}$ is a number of aggregate, accepting an input signal from the i-th communication channel, $1 \leqslant r_{i,j} \leqslant K$

$r_{i,2}$ is a number of input pole of the i-th aggregate, incidential to the i-th channel, $1 \leqslant r_{i,2} \leqslant N_{r_{i,1}}$

Outputs of the aggregates shall be defined by the matrix

$$H = \| h_{i,j} \| , \quad i = \overline{1,K}, \quad j = \overline{1, \max_{K}\{M_k\}} ,$$

where $h_{i,j}$ is a number of a channel incidental to the j-th output pole of the i-th aggregate.

Output signals of the aggregates shall be defined by the matrix $Y = \{ y_1, y_2, \ldots, y_L \}$, where $y_i = \{ \phi, \emptyset \}$, $\phi \neq \emptyset$

At any instant of time t_m^{SYS} we are supposed to know the moment for every aggregate, when there a sequential event can occur from the subset E'', i.e. we know the instants t''_{m_1} , $t''_{m_2}, \ldots, t''_{m_K}$ for which $t''_{m_k} \geqslant t_m^{SYS}$. Thus a sequential instant of time, when an event occurs in the system (i.e. the completion of one of the system processes) can be calculated as

$$t_{m+1}^{SYS} = \min_{1 \leqslant k \leqslant K} \{ t''_{m_K} \} \qquad (10)$$

Furthermore, as output signals of aggregates can be rearranged only at the instants of event occurence, then the instants of input signal arrivals in the aggregate are coincident with t_m^{SYS}

Taking into account the mentioned above, we are able to present the algorithm of aggregate functioning in the following sequence.

Step 1. Initial setting, when the instant t_1^{SYS}, t''_{1_1}, t''_{1_2}, ..., t''_{1_K} are defined.

Step 2. Estimation of t_{m+1}^{SYS} from the equation (1o).

Step 3. Aggregate access, for which $t''_{m_k} = t_{m+1}^{SYS}$, $1 < k < K$.

Step 4. Performing vector translations $\Theta(t_m)$ and defining output signals in accordance with (8) and (9). Calculation of the instant t_{m_k+1} from the equation (7).

Step 5. Matrix Y is looked through and if it has been found not an empty element $y_i = \phi_i$, then the $r_{i,1}$ aggregate access is rearranged; that is there comes a signal ϕ_i at its input pole $r_{i,2}$; transfer to step 4; but if not an empty element has not been found, transition will follow to step 2.

4. RESULTS OF THE SURVEY

According to above-mentioned principles, it has been constructed a program model, allowing to study the proposed structures.

The studies were conducted under following initial data of restriction.

1. The number of communication channels of the processor is assumed to be equal to 3 arrivals via channels to the host are independent of one another and the interval between them is exponentially distributed.

2. Job with certain degree of probability is among one of the ten classes, each of which is characterized by the values, that follow:

- input information content is 100 + 5000 bytes;
- output information content is 100 + 2000 bytes;
- labour-consumption per a job is 500 + 10000 oper./byte
- accepted response time is 200 + 86400 sec.

3. Host consists of 4 ten-partitioned computers; the processor speeds are respectively 10^6, 0.5×10^6, 0.1×10^6 oper. per sec.

4. Each job is channelled by a distributor into that computer, where expected processing time is minimal.

The results of modelling are tabulated in tables 1,2; for the first version they are as follows:

- speed of the processor for job distribution is V^{JD} = 20000 oper.per sec.

- speed of the SHI processor is V^{SHI} 1 200000 oper. per sec.

- specific volume of computations required to process information by SHI processor is equal to K^{PR} = 50 oper. per byte

- specific volume of computations, performed for the allocation of a job is σ^{JD} = 10000 oper.

For the 2nd version V^{JD} = 50000, V^{SHI} = 200000, K^{PR} = 150, σ^{JD} = 100000.

Table 1 shows mean statistical values of service times, where section 1 corresponds to the mentioned first class and section 10 to the tenth class; there are given in Table 2 utilization factors of processors.

Under given initial data the outcome of the simulation was the discovery of the fact, that all the three structures of the host are much alike in performance (i.e. the service time), provided lower rated of incoming flow, but taking into account processor loads, priority must be given to the first model; however, with greater intensities of incoming flow and moderate computations to allocate jobs (about 100000 oper.), the second structure appears to be more effective, for in this case the first structure has been liable to a marked queueing before the distribution system, that caused increased time-outs of the network channels.

And finally the third structure is to be used when large computations are performed for distribution (100000 oper. and more).

5. CONCLUSIONS

The studies, performed have shown, that efficient performance of multicomputer host within the distributed network is substantially dependent on the allocation of foreman services among the host computers. Applying the principles of program simulation, based on the theory of piecewise linear aggregates and the method of control sequences has demonstrated their efficiency in creating models of complicated computer system (computer complexity); this is because they allow to present the whole system as its individual interacting subsystems, for each of which a program model can be constructed by a standard technique.

REFERENCES

1. Ю.Б. Башин: Вопросы диспетчеризации информационно-вычислительных работ в ГСВЦ. Тезисы докладов к Международной научно-технической конференции ученых и специалистов стран-членов СЭВ и СФРЮ по проблеме: "Организация управления социалистическим промышленным производством". Серия № 6 "Техника и технология управления", часть II, М., 1975.

2. Ю.Б. Башин, В.А. Ильин: Информационные сети с ЭВМ. Знание, М., 1976.

3. L.Kleinrock: Analytic and simulation methods in computer design, SSCC, 1970, pp.569-579.

4. Ю.П. Зайченко: Исследование операций. Издательское объединение "Виша школа", Головное издательство, Киев, 1975.

5. Н.П. Бусленко, В.В. Калашников, И.Н. Коваленко: Лекции по теории сложных систем. М., "Советское радио", 1973.

6. В.Н. Бусленко, Н.П. Бусленко, В.В. Калашников, В.И. Лутков: Имитационная модель агрегативной системы. Программирование, 1975, № 1, М., "Наука", стр. 60-71.

7. А.И. Волков, П.П. Живаткаускас, Г.И. Праневичюс: Применение метода управляющих последовательностей для описания функционирования мультипрограммной ЭВМ. Материалы конференции "Развитие технических наук в республике и использование их результатов", Каунас , 1975, стр. 11-19.

Y. Bashin, I. Gorelik, H. Pranevitchius
Kaunas Politechnical Institute
Lithuana
USSR

TABLE 1

VARIANT	MODEL	TOTAL RATE OF REQUEST (REQ./SEC)					
		=0.06		=0.15		=0.18	
		SEC.1	SEC.10	SEC.1	SEC.10	SEC.1	SEC.10
I	1	2.68	82.7	3.04	139.1	3.76	175.0
	2	2.76	81.4	3.28	148.6	3.54	169.8
	3	3.15	83.0	3.82	140.0	4.05	178.2
II	1	3.92	90.0	6.85	142.1	7.36	177.3
	2	3.34	88.2	5.58	146.0	5.85	173.0
	3	4.12	92.1	8.68	147.4	8.95	178.6

TABLE 2

VARIANT	MODEL	UTILIZATION FACTOR OF EQUIPMENT					
		CPU_1	CPU_2	CPU_3	CPU_4	CP	SP
I	1	0.715	0.613	0.562	0.0	0.131	-
	2	0.799	0.618	0.230	0.199	0.118	-
	3	0.756	0.698	0.327	0.001	0.117	0.075
II	1*	0.747	0.646	0.365	0.0	0.428	-
	2	0.752	0.659	0.454	0.0	0.352	-
	3	0.752	0.629	0.453	0.0	0.353	0.3

* in this case a considerable increasing of queue for scheduling requests occurs

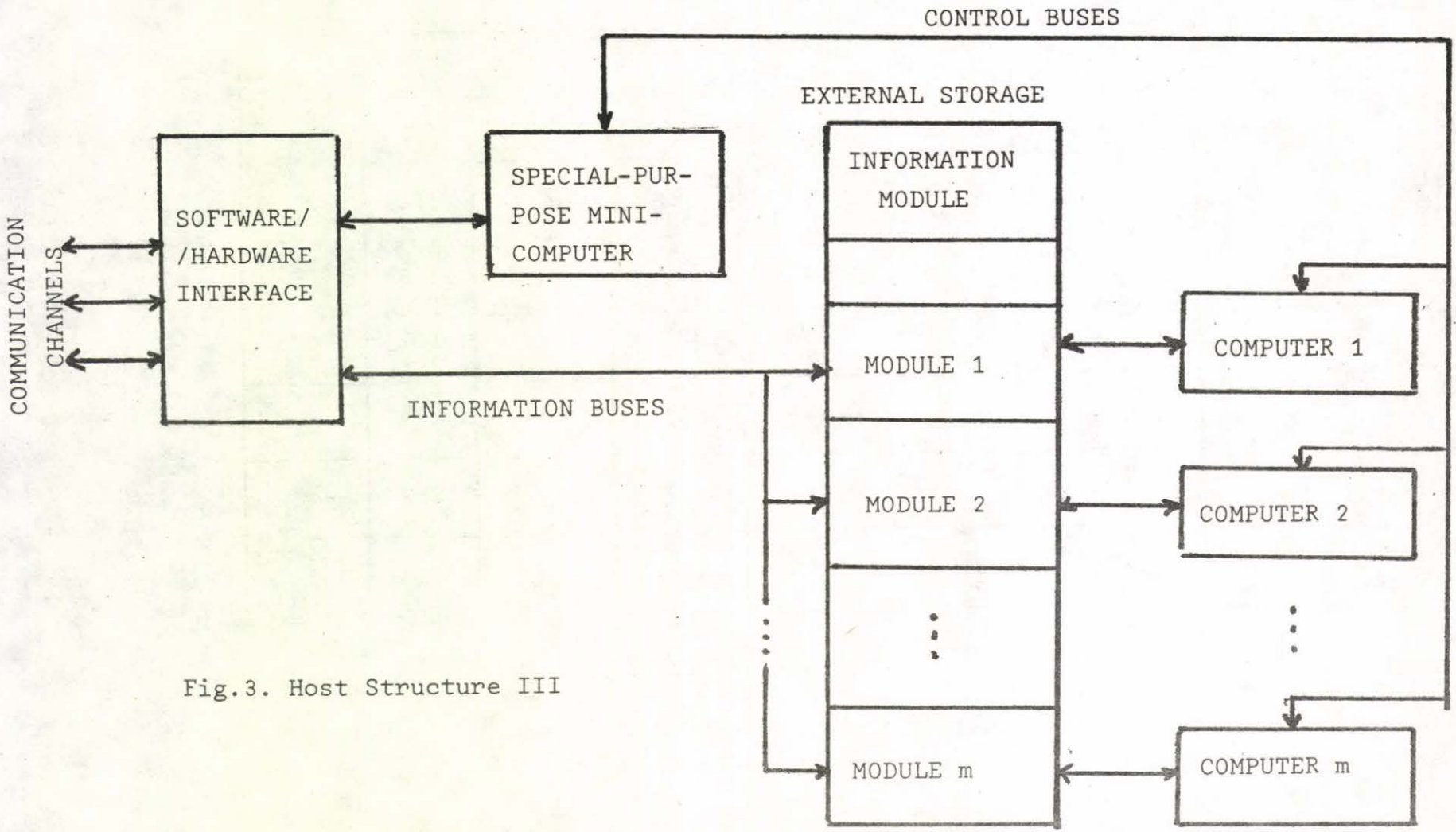


Fig.3. Host Structure III

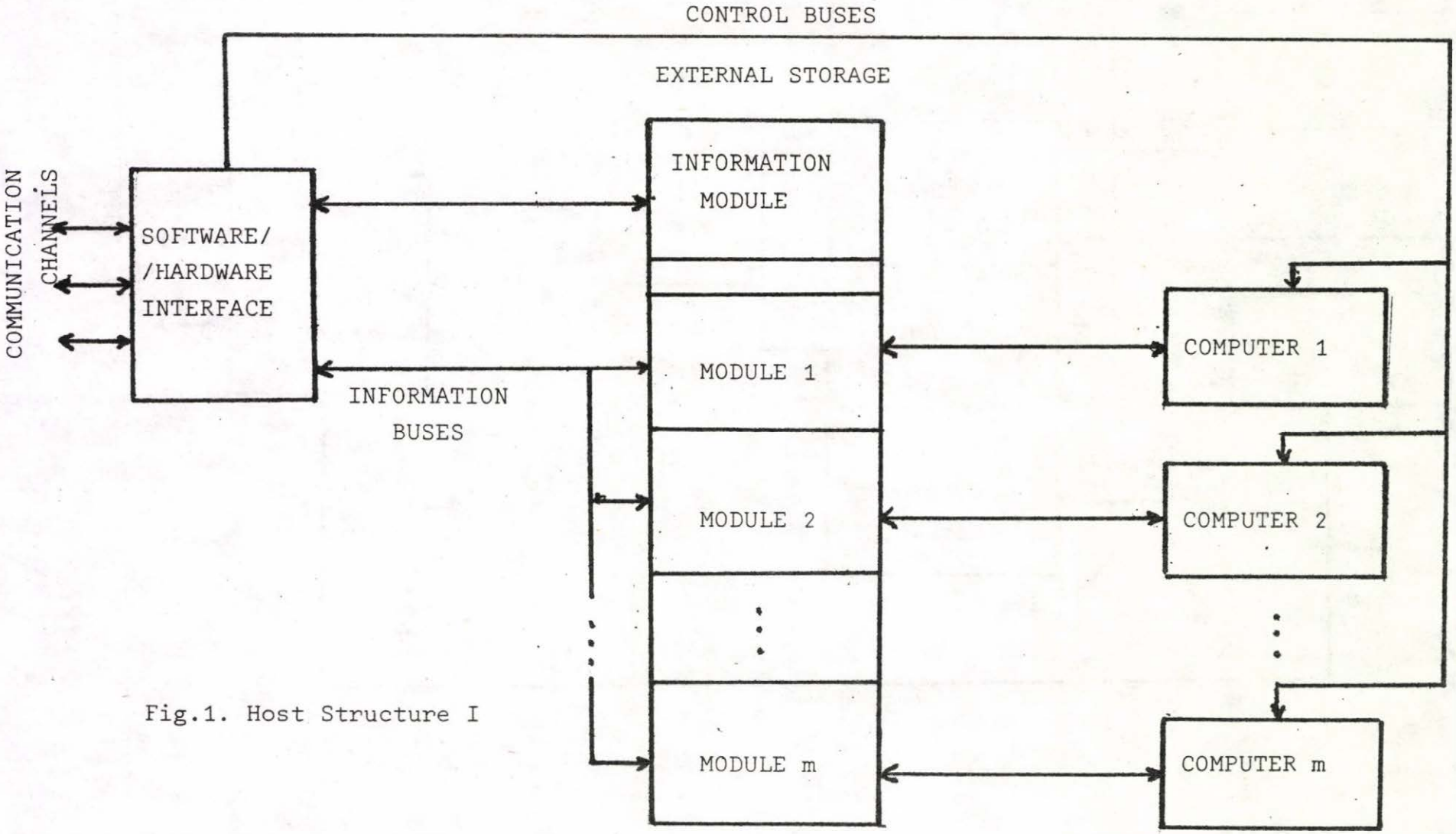


Fig.1. Host Structure I

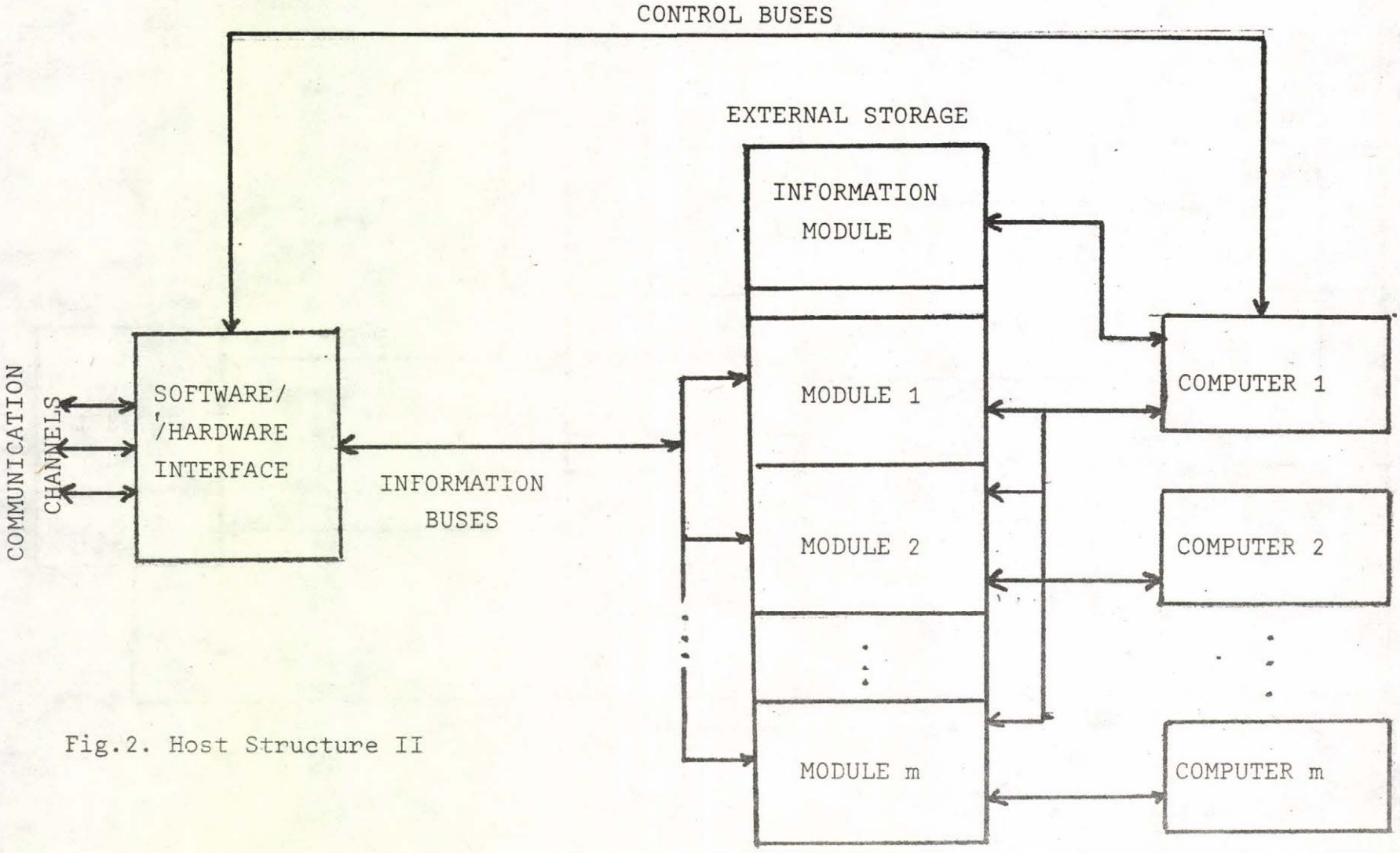


Fig.2. Host Structure II

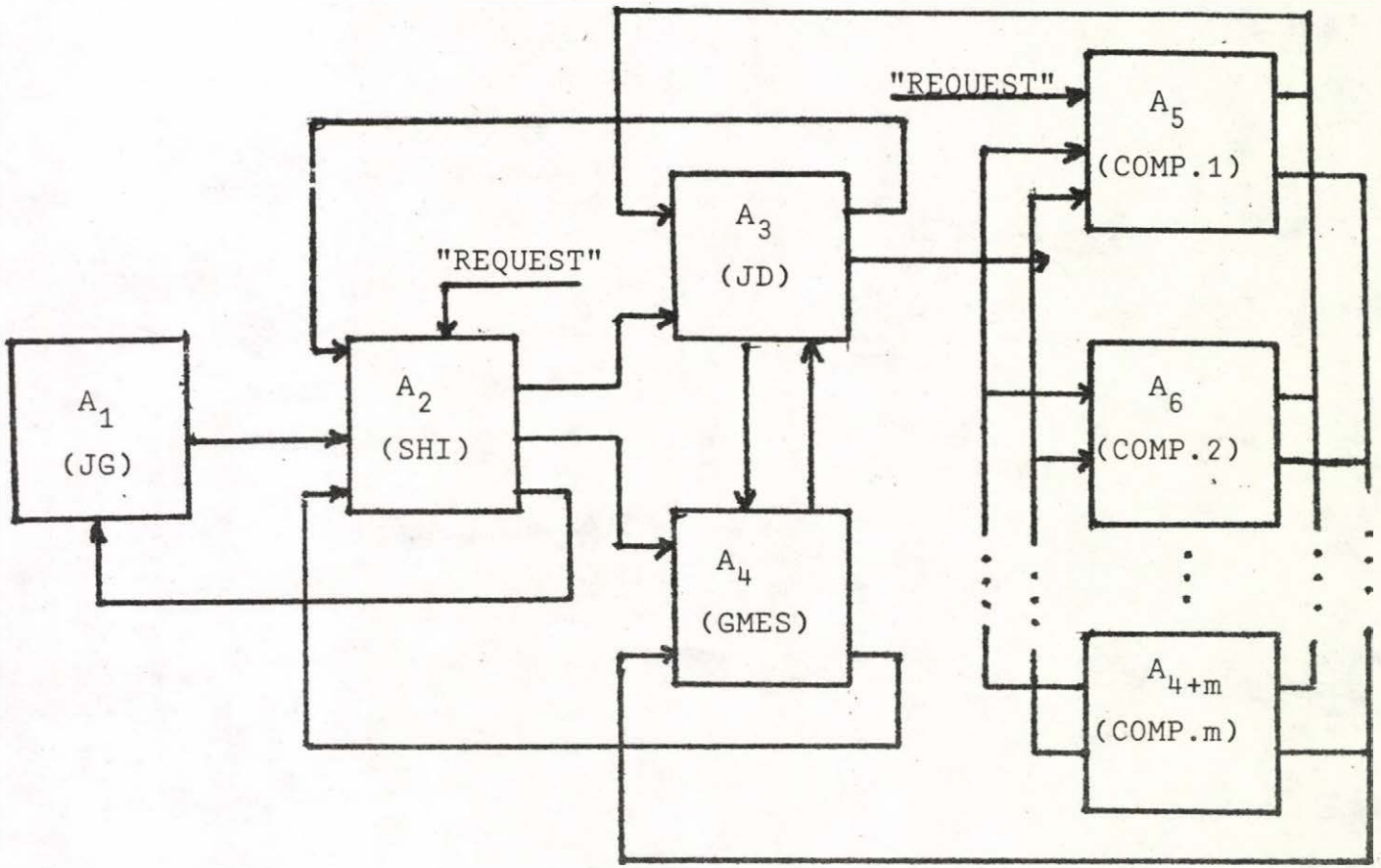


Fig.4. The Aggregate System of Host

AN INTEL 8080 BASED MULTITASK SUPERVISOR

Peter Bernus

Hungary

Abstract

The multitask environment maintained by the supervisor described is intended for application in realtime controllers which have to perform more or less independent functions and a convenient way to write the controller's program is the multitask approach.

The paper presents how the structured analysis method SADT[R] has improved the design and the implementation of the system.

[R] SADT is a Trade Mark of Sof Tech Inc. Waltham, Mass. USA

Requirements

This Multitask Supervisor /MTSV/ was intended to be used in realtime controllers, built up on Intel 8o8o base - where a convenient approach to write the controller's program is the multitasks one.
A typical task is for instance to control a material handling subsystem of an Integrated Manufacturing System. Parallel tasks can be for instance:

- to control a mechanical palet changer equipment through a serial I/O line
- to perform a dialog with a worker through keyboard and alphanumeric display
- to send sometimes status reports to a central computer
- to control the sequence of other slow mechanical equipment.

The requirements such a supervisor should meet are listed on Fig.1. The MTSV should be an independent software component, so that you can run your machine code level programs /here called tasks/ on both dedicated hardware and on a hardware shared between tasks by the help of the MTSV. The Second independency requirement is the independency of the SV from I/O, resulting in the transportability of the SV between different microcomputer configurations. The SV has to support the distribution of tasks between processors i.e. when the programmer is writing his tasks, he or she is not forced to keep in mind on which actual processor his or her task will run. Thus, hardware interfaces should remain transparent to the task-communication. The only thing to do

when a task-communication queue crosses the processor's bounds is to break the line there which automatically inserts a queue-linking interface. /Figures 2 and 3/

The last independency requirement as stated on figure 1. was the independency of task - I/O from hard-I/O. Therefore I/O tasks work on and/or from I/O queues while they change information with IT routines via dedicated tables.

Solution

The diagramming technique SADT^{R} was used to determine the functions the MTSV has to perform.
The first decomposition /fig.4./ shows the functions on one processor. The main and only task of the SV is to administer tasks. For this purpose it uses or accesses

- STATUS and STACK information
- a TASKTABLE
- TIMER interrupts
- QUEUES' STATUS

The ADMINISTER TASK function is divided into subfunctions in a perfect and disjunt way as shown on fig.5. Arrows representing those and only those data /-classes/ enter the block of ADMINISTER TASKS, which did it on its parent diagram A $\emptyset$. There is a similar rigorous correspondence between controlling data and output data respectively. A further decomposition of block No.2. of this actigram is shown on figure 6. This is the actigram of the function "SCHEDULE".

Block No.1. on this figure - which is seen to be activated by IT1 or a simulated interrupt - represents the first action to be performed by the MTSV. It is clearly shown

that the current status is saved onto the stack /being part of the stacks/ and the actual task-state and stackpointer information are saved into the task-table.
The function makes also use of the pointer depicting the current tasktable-entry.

Characteristics /summarized on table 7./

The MTSV supposes the presence of an interrupting hardware timer device which clocks every 1 msecond and enables the supervisor to change tasks continuously. The same clock is used for measuring the elapsed time of "sleeping tasks".
Another timer with 10 msec. interval is used for the maintenance of a realtime-register. Every task can use an alarm-clock of his own and be woken up at the desired time by the MTSV.
Information flow between tasks and hardware as well as intertask communication is maintained via message queues. A common, uniform queue area handling subroutine package is the base stratum of the MTSV. However this task-turnaround administration is made through additional tasktable. This is due to time considerations. When using a 1 msec timer and running 8 parallel tasks on an Intel 8080 the MTSV overhead is about 27 p.c.

An actual system's generation requires the addition of peripheral driver subroutines to the MTSV, which are supplied from output queues or supply input queues with input data respectively. Every task has a queue--number /or can have it/ and a task number. The queue-number, if any, must be equal to the task number.

Every task has a dedicated stack area used for subroutine

calls and parameter passing as well as temporary data storage.

Tasks can put 16 bit data into and get from any queue. A task can require the MTSV to let him sleep until it's input queue /or another specified one/ is not empty.

The states a task can have under MTSV are shown on fig.8 together with possible transitions. Transitions are initiated either by the MTSV or through MTSV primitives.

MTSV primitives:

STOP; calling task stops.

STOPTS n ; stops n'th task except if it is the running one /itself/

ABORTS n ; aborts n'th task. The current value of the stackpointer of the aborted task is saved into the tasktable.

ABORT ; aborts calling task.

WCY; makes the calling task wait a task-turnaround time.

RUNTS n ; if the n'th task was in "STOPPD" state, it will enter the WCY state.

WY T ; the calling task enters the WY state and will be woken up after T milliseconds will have elapsed.

WRT LOC RT ; the calling task enters the WRT state e.g. it waits until the mask-value referred by the parameter RT of the primitive call becomes smaller than the value of the MTSV's realtime register.

WQ n ; the calling task enters the WQ state and will be woken up if the n'th queue is not empty. Typically

a task waits for his own input queue. If in the moment the task calls WQ the n'th queue is not empty the WQ n call gives control back to the calling task without any delay.

WHOAMY n; this is a feature of the MTSV which enables the programmer to write reentrant subroutines which are parallelly used by several tasks. This is the way such a subroutine can decide under which task he is actually running.

GIVERT LOC RT ; the MTSV copies the contents of the realtime registers into the area pointed to by the primitive call's RT parameter.

The areas under the MTSV's responsibility are shown on fig.9. These are

- Realtime registers
- tasktable
- current tasktable entry /pointer/

As mentioned earlier there is a stratum between the hardware level and the MTSV. This is the queue area handling one /fig.lo/./We don't deal here with the IT routines./

Queue area handling

The queue area handling can be considered from two aspects. The one is as the free area management sees it /fig.11/. Here, there is a double pointer free area chaining/handling. This is the base of the fact, that the system uses only one common queue area which is used as a resource of memory cells. Due to this only the sum of the length of the queues is limited.

The other aspect /fig.12/ of the queue area handling

is as the queue routines see it.

There exists an array of queue entry /IN/ pointers and another for queue-end pointers /OUT/. The first is used by the GETQ , the second by the PUTQ routines. Queue routines - being common and not shareable - run under IT disable. The pessimum of running times is given on the figure.

Conclusion

A multitask supervisor for use in realtime controllers without extreme requirements in speed has been presented. Special attention was given to the structured design and analysis method which enabled us to design and implement this MTSV within as short as 3-4 weeks including design, coding and debugging. The relatively low evaluation time has proven the efficiency of the method and convinced us of the necessity using structured methods for specifying and coding. However coding was done in assembly language the longest part of the code which has not been designed in SADT before was about 5o bytes long.

- TABLE 1 -

MTSV: MULTITASK SUPERVISOR

REQUIREMENTS:

THE MTSV SHOULD BE INDEPENDENT OF OTHER SYS COMPONENTS

- MACHINE CODE LEVEL PROGRAMS ARE TO RUN ON BOTH DEDICATED AND MULTI--HARDWARE
- I/O AND MULTITASK HAS TO BE INDEPENDENT

. THE MTSV IS CONCERNED WITH A TOOL ENABLING US TO ASSIGN PARALLEL TASKS TO A SET OF DISTRIBUTED PROCESSORS.

. INTERTASK COMMUNICATION AND I/O SHOULD BE UNIFORM

. I/O TASKS WORK ON AND/OR FROM QUEUES AND CHANGE INFO WITH IT ROUTINES VIA DEDICATED TABLES

QUEUE LINKING INTERFACE

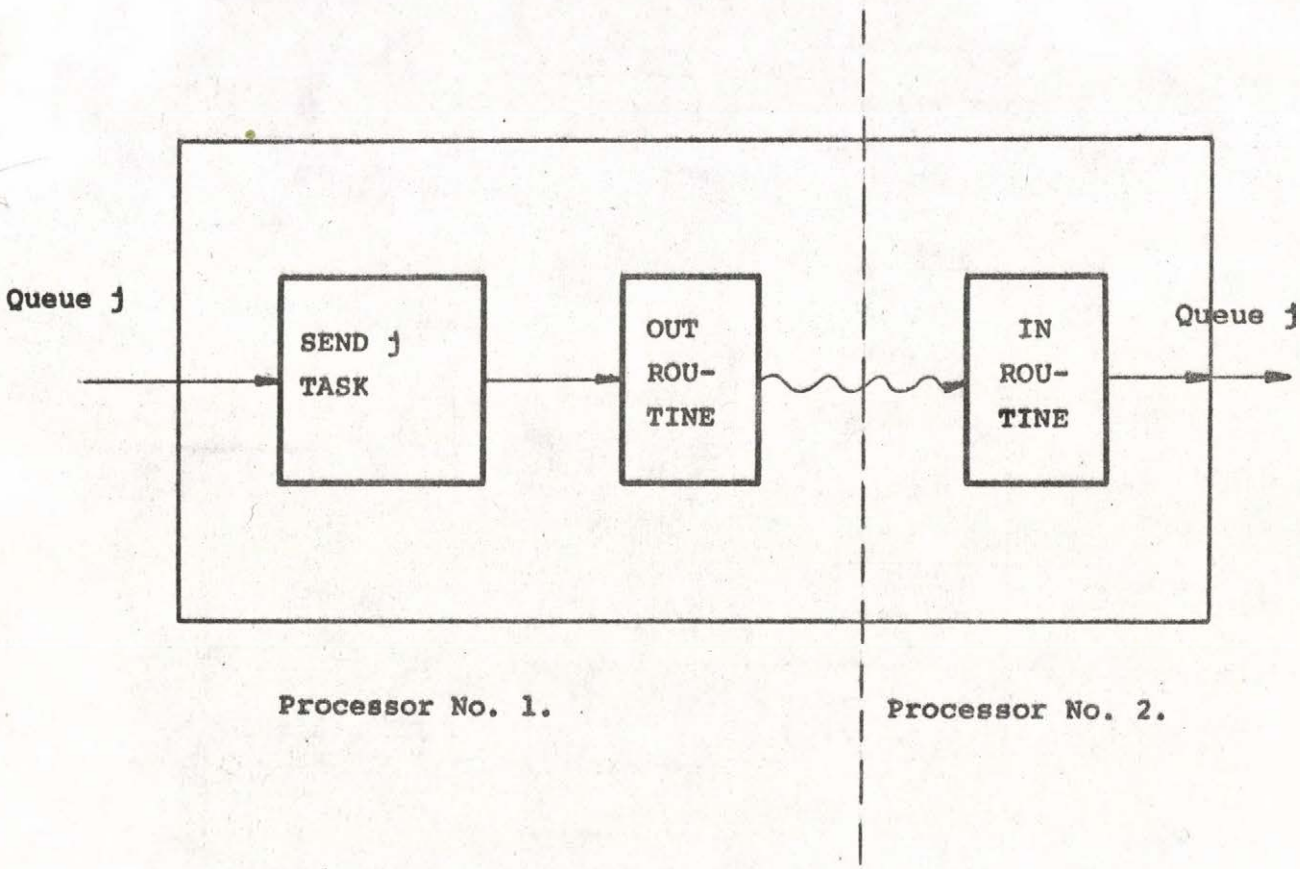


Figure 3

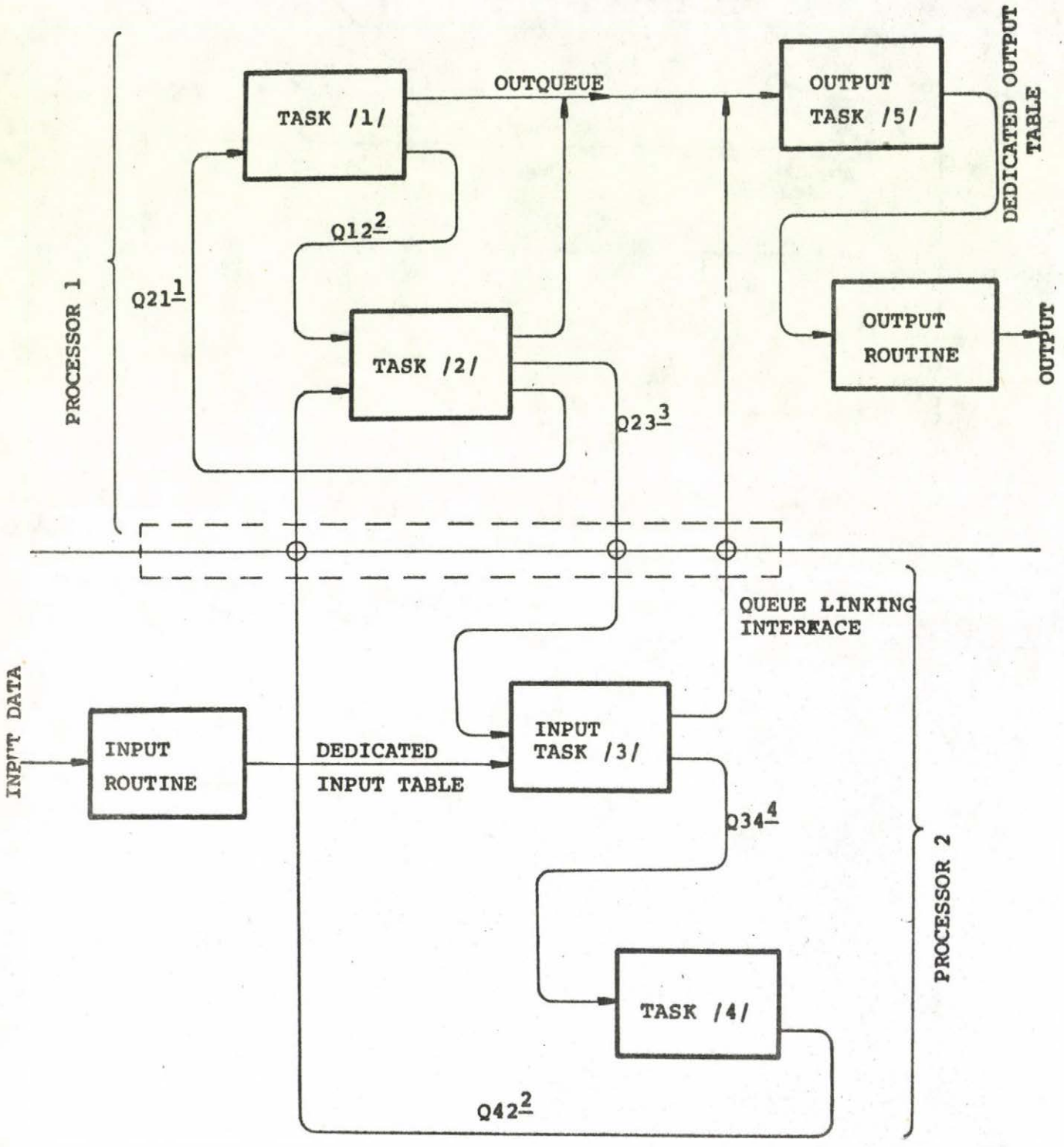
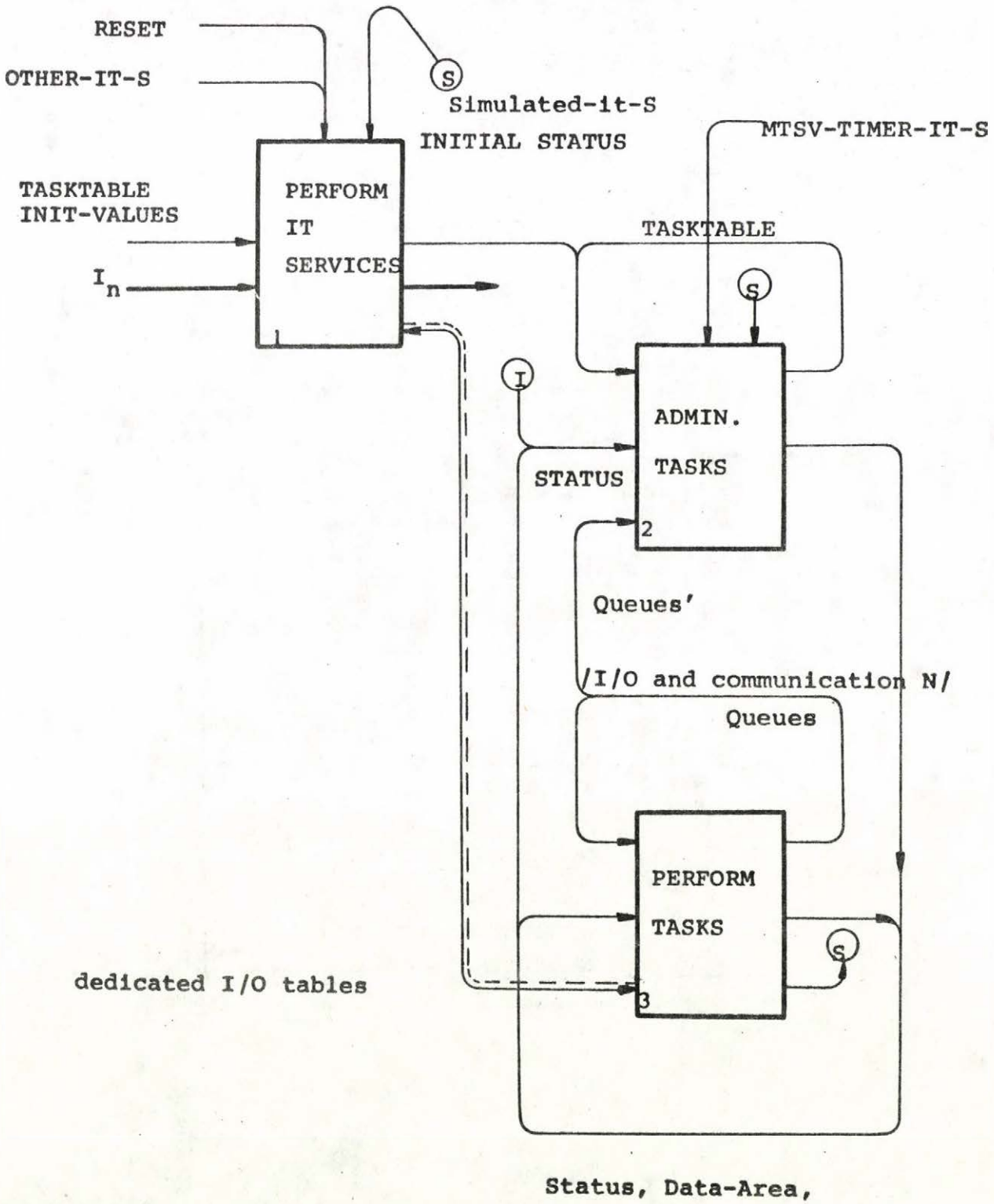

ALLOCATE TASKS TO PROCESSORS
PROCESSOR 1
TASK /1/
OUTQUEUE
OUTPUT TASK /5/
DEDICATED OUTPUT TABLE
Q12 2
Q21 1
TASK /2/
OUTPUT ROUTINE
OUTPUT
Q23 3
QUEUE LINKING INTERFACE
INPUT DATA
INPUT ROUTINE
DEDICATED INPUT TABLE
INPUT TASK /3/
Q34 4
PROCESSOR 2
TASK /4/
Q42 2

Figure 2

SATD[TM] AØ ACTIGRAM OF THE MTSV RUNNING ON ONE PROCESSOR



TM of Softech. Waltham, Mass. USA

Figure 4

MTSV-A2 ACTIGRAM ADMINISTER TASKS

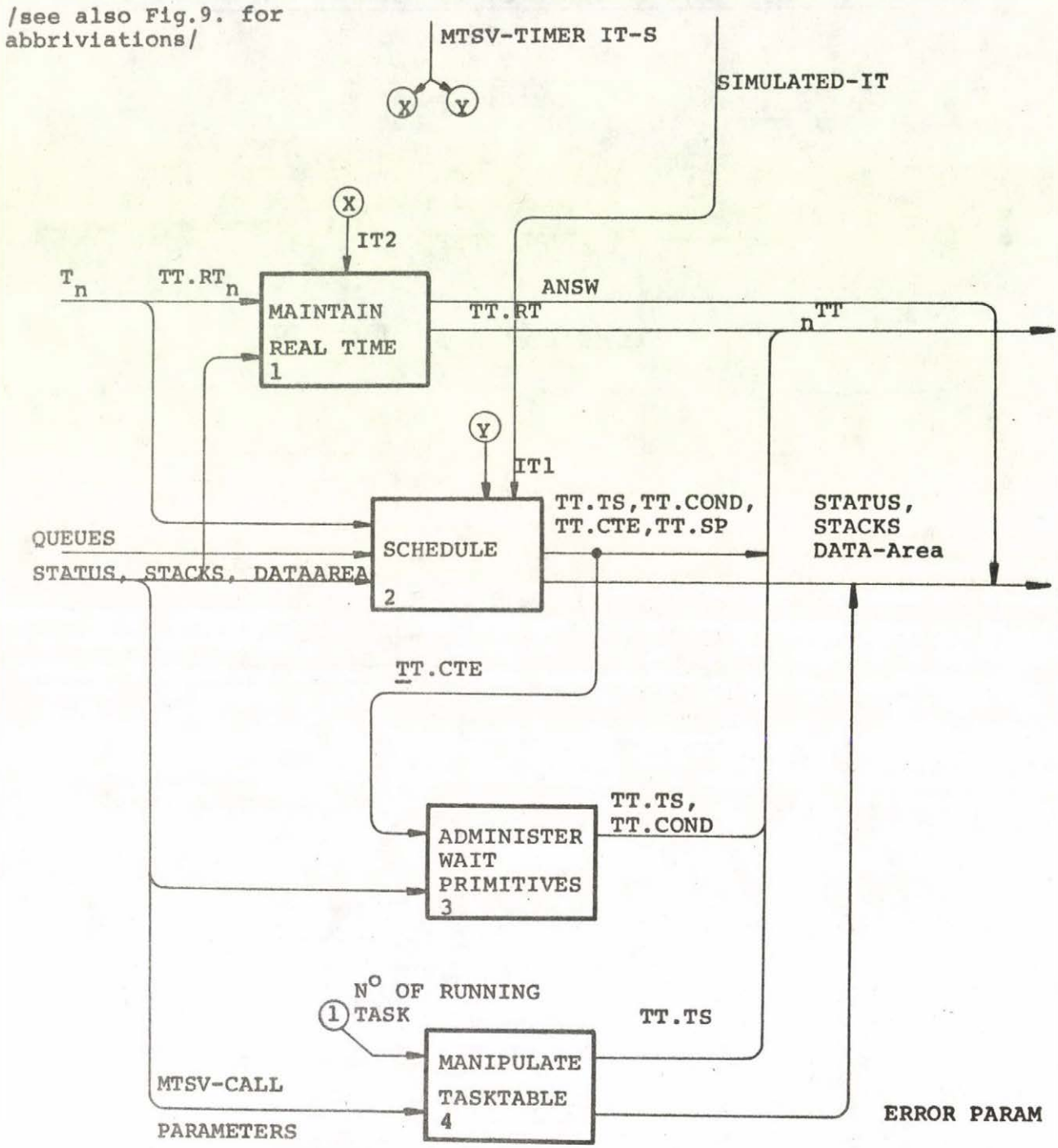


Figure 5

MTSV - A2.2 ACTIGRAM SCHEDULE /see also Fig.9. for abbreviations/

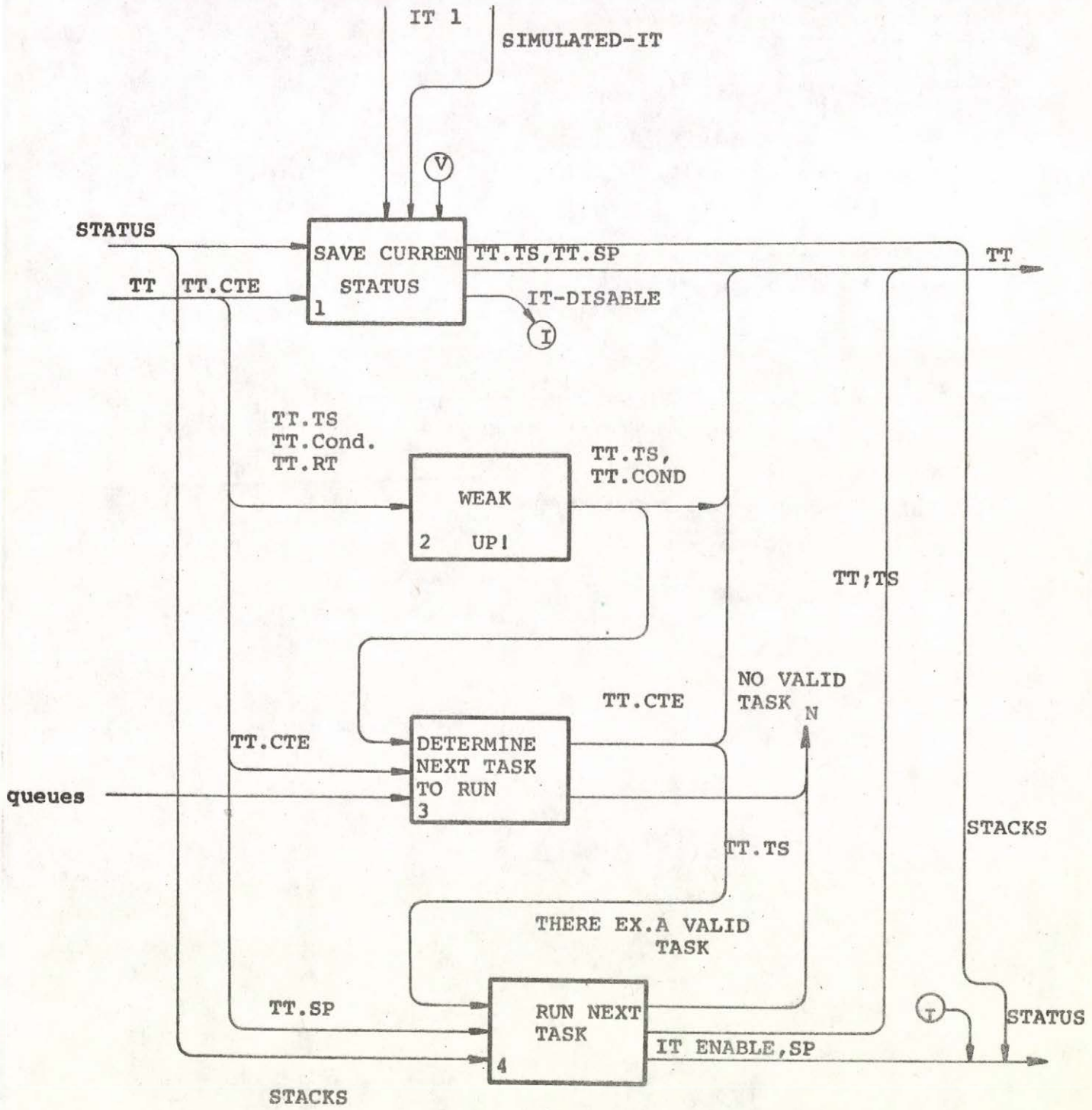


Figure 6

- TABLE 7 -

CHARACTERISTICS

. 1 msec timer, change tasks
sleeping tasks
. 10 msec timer, realtime register

. message queues
. queue handling package
. tasktable.

ex:
INTEL 8080, 8 tasks, 1 msec timer
. overhead : 27 p.c.
. task/s/: dedicated input queues
. Application area - small
- sophisticated control
- no strong time requirements

TASK-STATES AND STATE TRANSITIONS

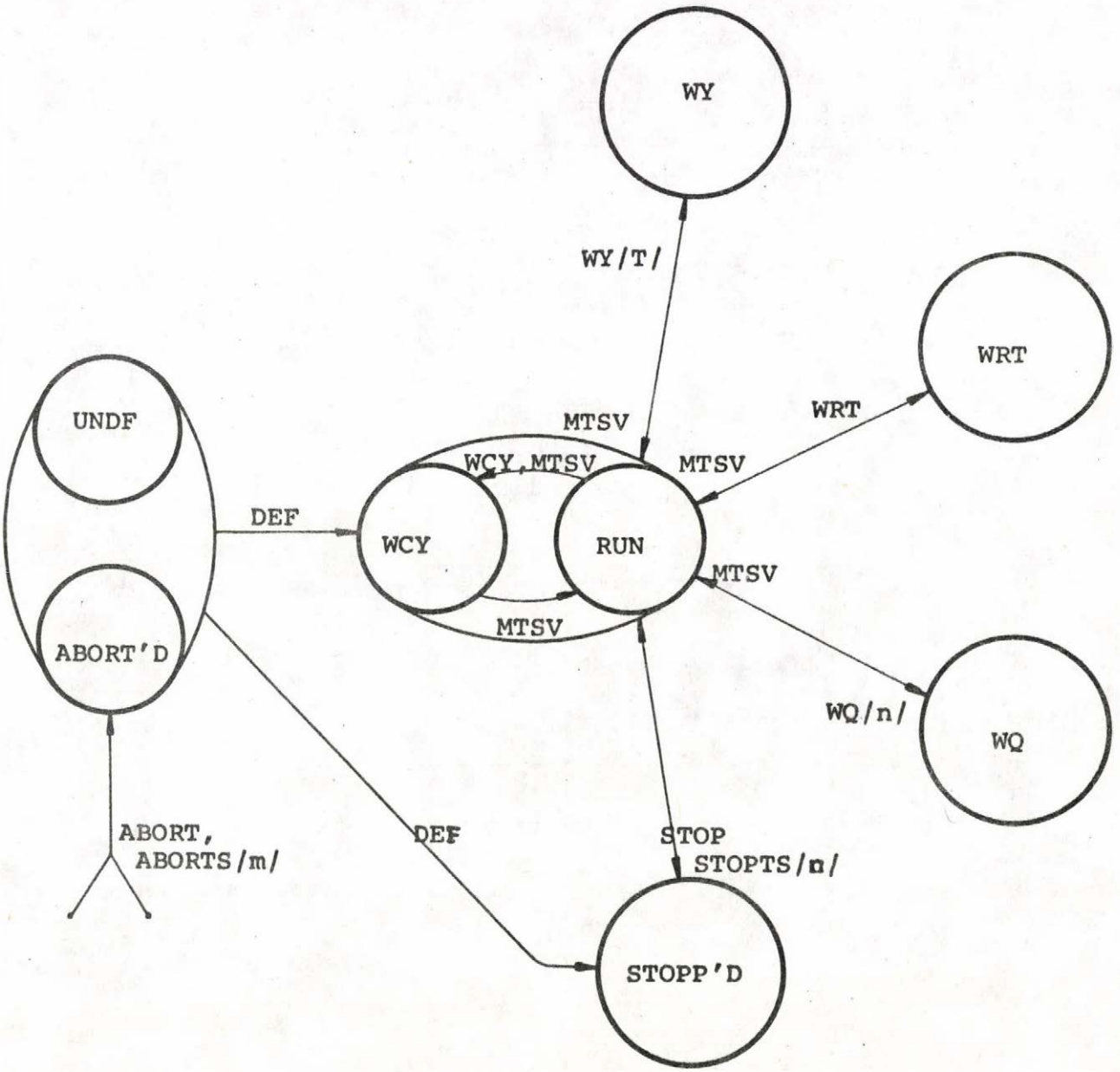
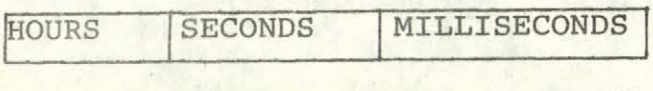


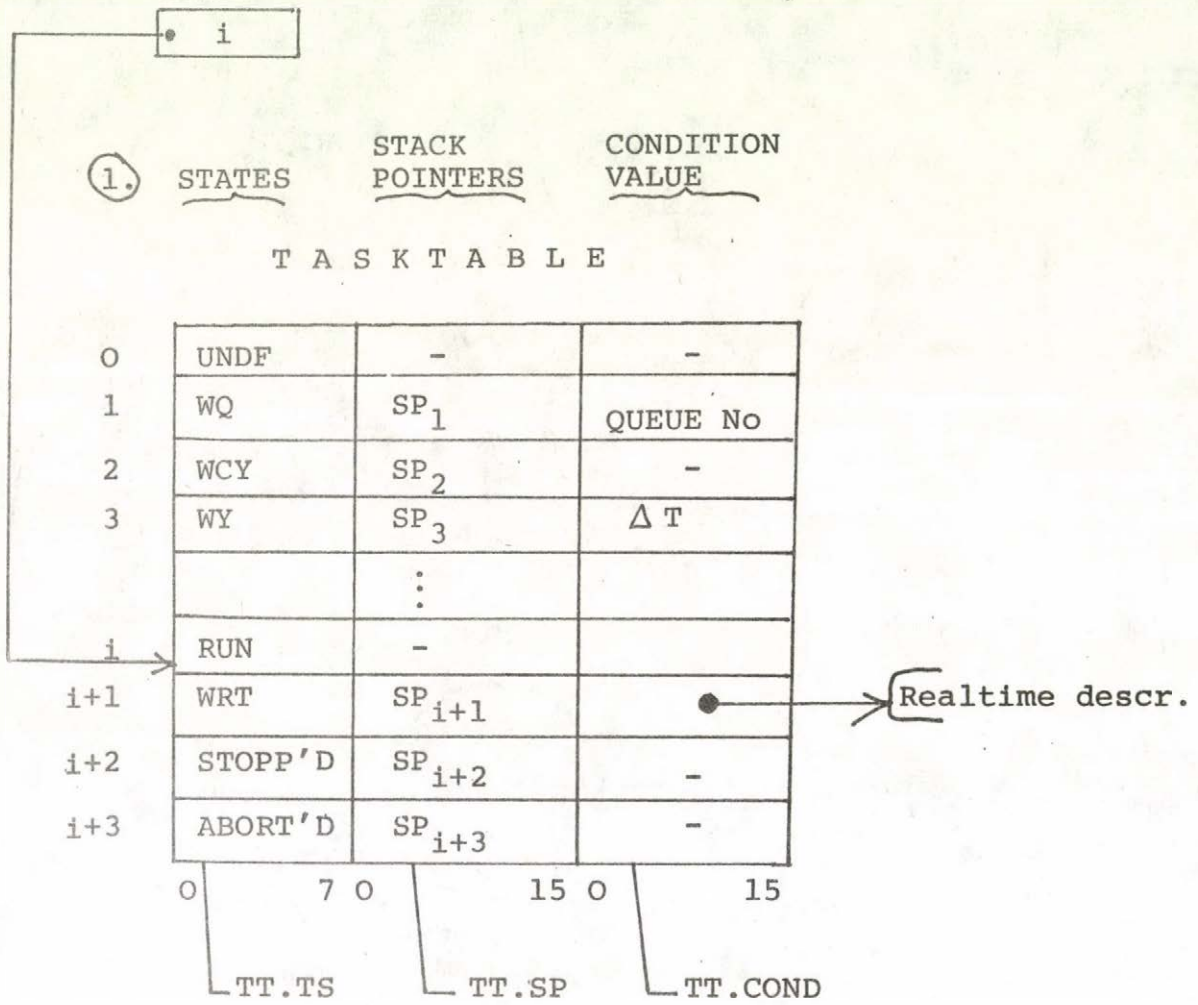
Figure 8

AREAS WITHIN THE MTSV'S RESPONSIBILITY

REALTIME REGISTER(TT.RT)



CURRENT TASKTABLE ENTRY (TT.CTE)



- Fig.9 -

MTSV STRATA

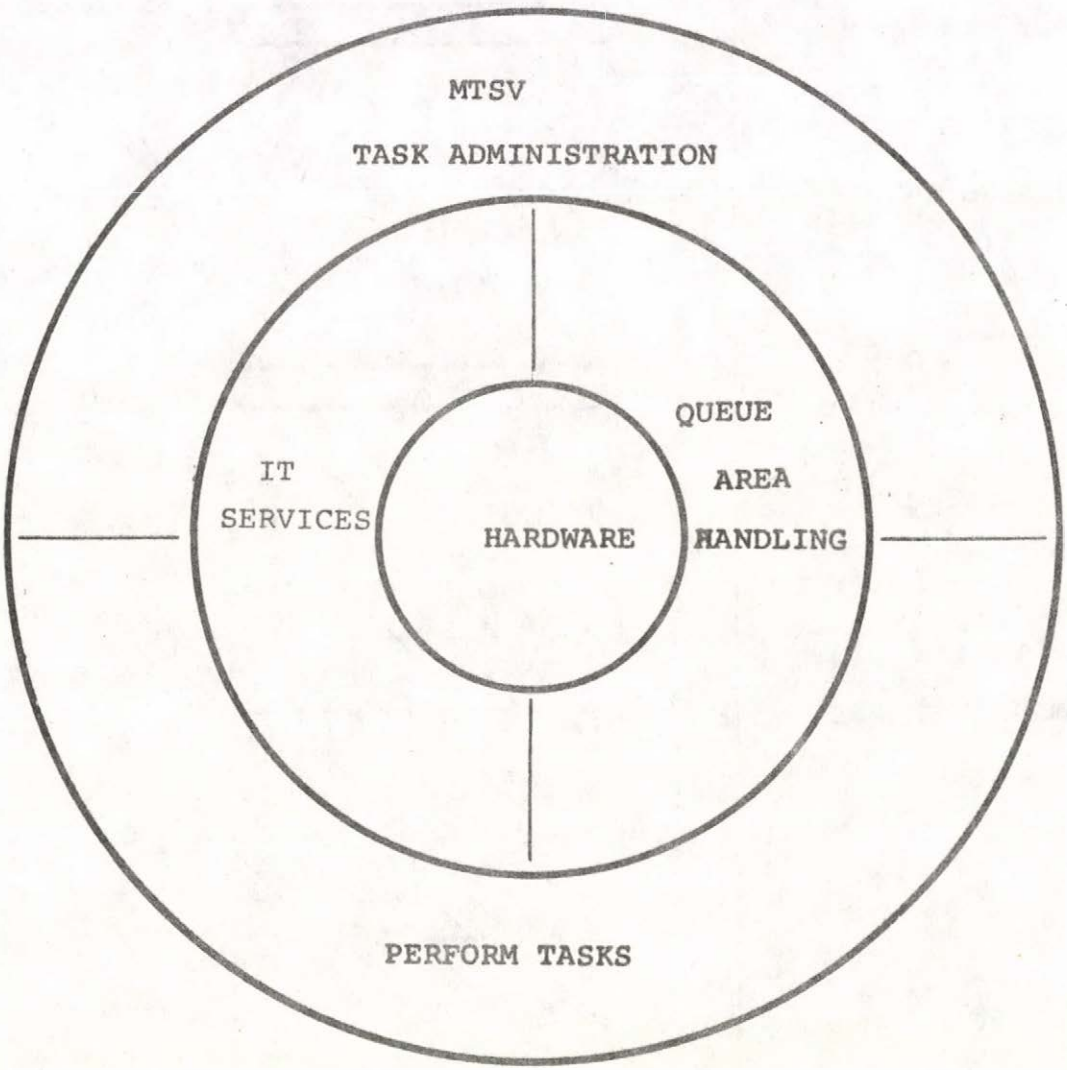


Figure 10

QUEUE AREA HANDLING

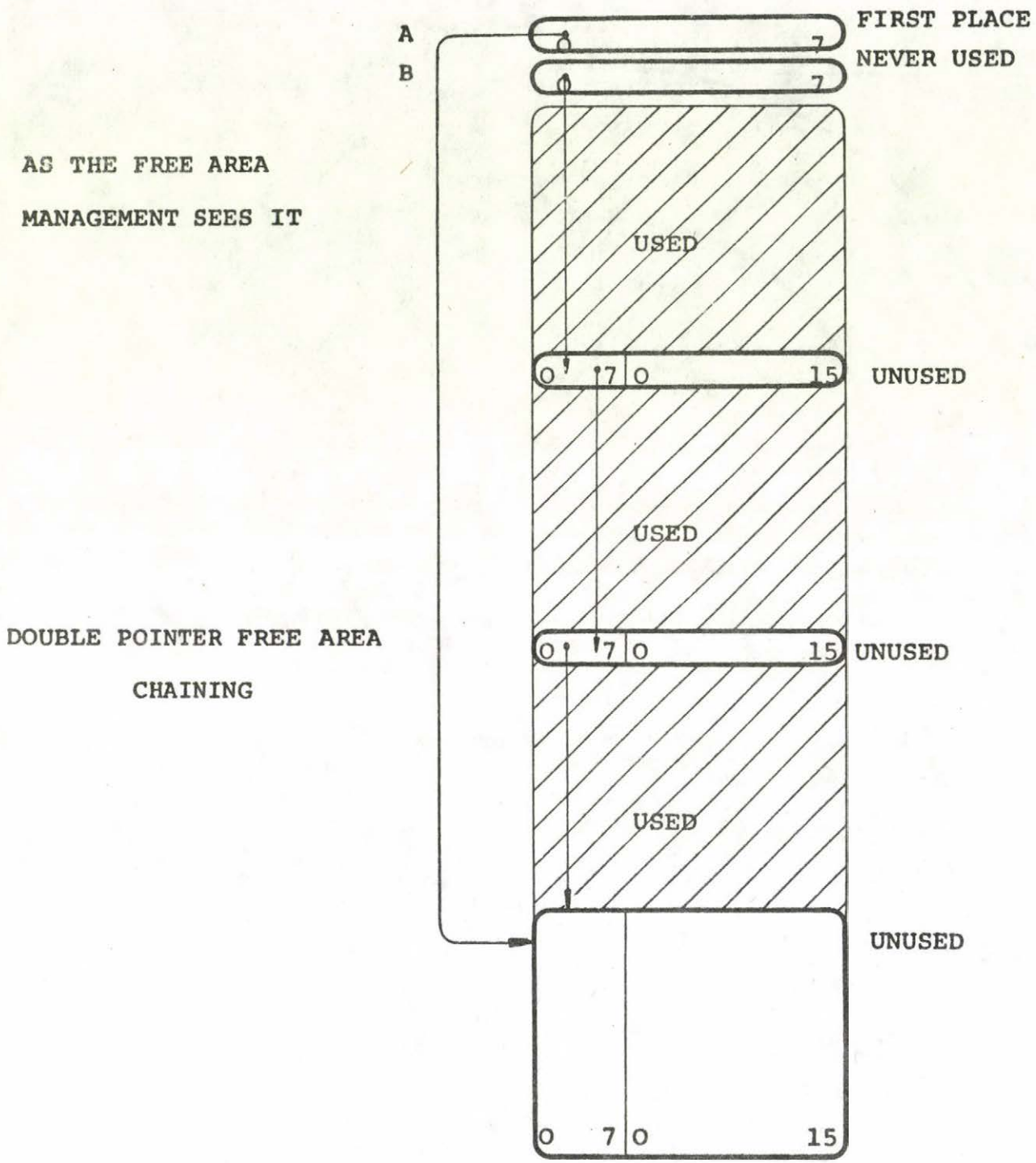
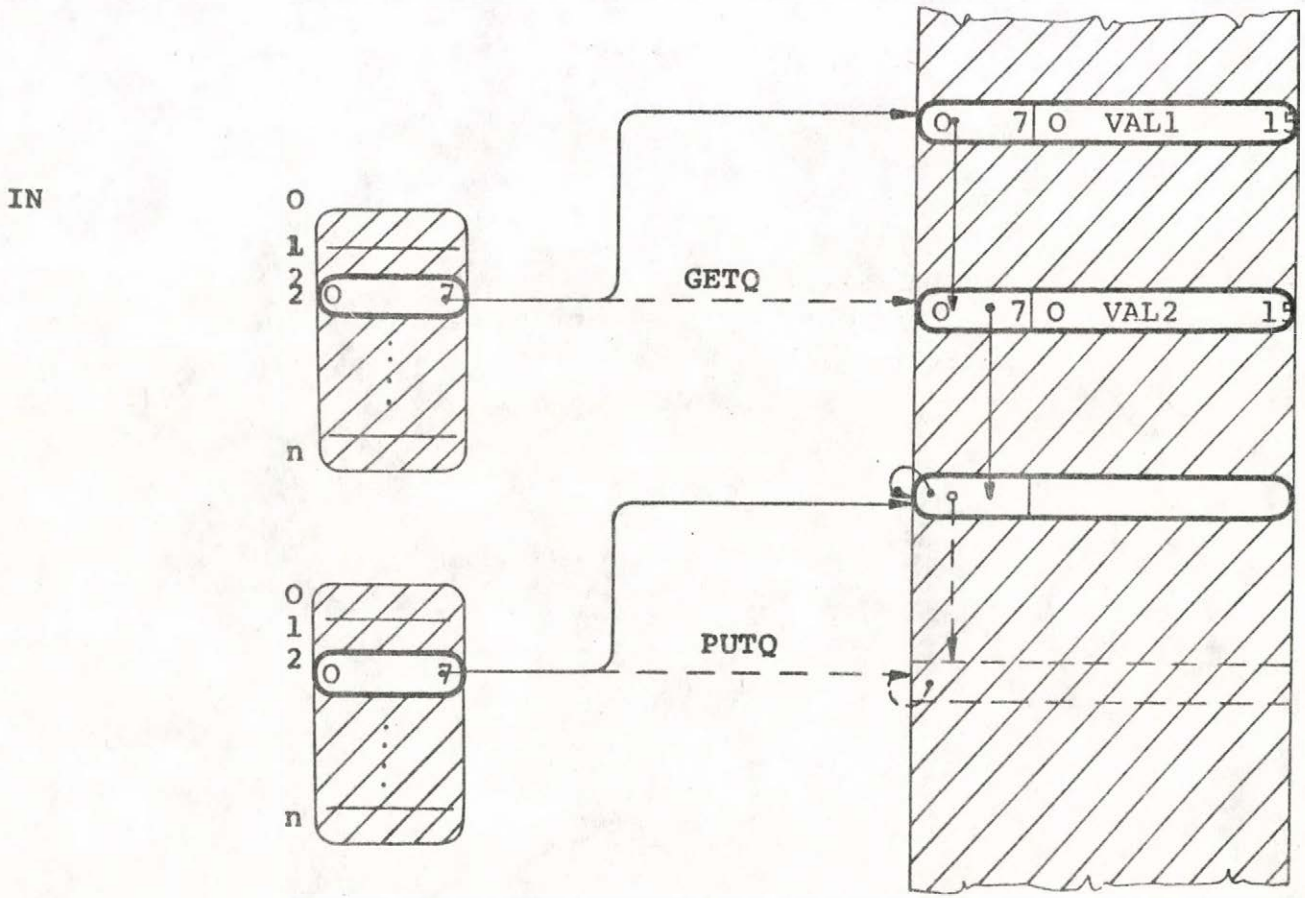


Figure 11

QUEUE AREA HANDLING

/AS THE QUEUE ROUTINES SEE
IT WHEN HANDLING QUEUE N°2/



QUINI 850 µs
PUTQ < 380 µs
GETQ < 380 µs
ASKQ 140 µs

Figure 12

PROBLEMS AND SOLUTIONS FOR UPDATING MULTIPLE AND DISTRIBUTED COPIES OF DATA *

by

Georges GARDARIN **
INSTITUT DE PROGRAMMATION
UNIVERSITE DE PARIS VI

ABSTRACT

Several concurrency control methods for updating partially redundant distributed databases have been proposed. Each method is generally introduced in its own context and takes care of different problems as interferences, deadlock, robustness to site crashes and network partitions. This paper first introduces a unique context and then presents an overview of seven algorithms and methods. The centralized locking algorithm is straightforward, but it needs a sophisticated back-up strategy. Decentralized locking algorithms allow to increase the reliability of the distributed system. Between centralization and decentralization, the primary site methodology appears as a compromise. The decentralized voting algorithm looks like a challenge toward introducing independence between sites. Finally, the decentralized transaction ordering algorithms and the preanalysis method can be seen as attempts toward simplicity.

KEY WORDS AND PHRASES

Distributed databases, redundancy, concurrency, crash recovery, consistency

---

* This work is sponsored by IRIA SIRIUS and SEFI

** Current address: Computer Science Depatment BH3531
University of California LOS ANGELES , CA 90024
USA

1. INTRODUCTION

Updating redundant databases in a distributed system needs devlopments of specific algorithms in order to control the untoward effects of concurrency. A lot of proposals have been suggested during the last years. The current paper presents a survey of the main contributions in this area.

The problem which is faced is called interferences (TRIN75). Given two or more independent transactions, an interference is said to have occurred if the result produced by their concurrent execution would not have been obtained by running these transactions one at a time in any order (BACH77). Interferences are involved by sharing data among concurrent processes; in the following, any portion of the database located on one unique site controlled by the distributed system is called a resource.

Interferences existed before the advent of distributed processing. In centralized databases like in distributed one, they can be classified in two types (GARD77) :

- lost operation which appears when the result of an operation is destroyed by another operation executed at the same time on the same resource,
- inconsistency which may occur when a transitory inconsistent state of the database is printed or modified by a transaction, or when a transaction performs two non reproducible reads of the same data (ENGL76).

Of course, in distributed databases, lost operation may occur only locally where a resource is. However, there are two kinds of inconsistencies :

- local one which involves local integrity constraints, that is to say assertions which must be satisfied by two or more resources located on the same site,
- distributed one which involves distributed integrity constraints, that is to say assertions which must be satisfied by two or more resources located on different sites; a special case of distributed integrity constraint is duplication of resources, which is also known as the redundancy case or the multiple copy one; this case is of highest importance in distributed systems in order to ensure reliability of the system (DAVE78).

Moreover, in a distributed system, the updating of a resource needs additive time in order to perform the network communication and to update the multiple copies. Futhermore, the volume of transactions may be important because computer facilities are generally numerous. Finally, the frequency of interferences is expected to be higher than in centralized database and the resolution of interferences can appear as the first problem which must be solved in order to built distributed data management systems (FRY75, ADIB78).

In the following, various proposed solutions are analysed on a pratical point of view. They are described with the help of a common view of distributed data management systems. This view, which is developed in the second section, allows to define a unique way to describe algorithms generally intoduced in different contexts. In the next sections, the algorithms are presented. First of all, the centralized locking algorithm (ELLI77) is straightforward, but it needs a sophisticated back-up strategy (MENA78). Then, two variants of a decentralized locking algorithm (ELLI77, CHU74) allow to increase the reliability of the distributed system. Between centralization and decentralization, the primary site methodology (ALSB76, STON78)

appears as a compromise. Then, the decentralized voting algorithm (THOM76) looks like a challenge toward introducing independence between sites. Finally the decentralized transactions ordering algorithms (GARD78, ROSE78) and the preanalysis method (BERN77) can be seen as attempts to simplify the system implementation.

II. DISTRIBUTED DATA SHARING MODEL

A distributed database management system is assumed to include four ingredients :

- a set of computers called sites,
- a data communication network which allows and controls the transmission of data between the sites,
- a set of databases, each database being partitionned and partially duplicated over several sites,
- an infinite set of potential transactions, each of them corresponding to the execution of a user program located on a unique site.

Each database is divided in a collection of data units, each of them being localized on one and only one site. The transactions share these data units which are called resources. The whole set of resources which belong to the same database and are managed by the same site is called a local database. Each local database must obey to a set of integrity constraints called the local integrity constraint. All the local integrity constraints are supposed to be implicit, that is to say unknown by the distributed system. Moreover, it exists some integrity constraints between the local databases which compound a distributed database. There are two kinds of such distributed integrity constraints :

- implicit ones which are not known by the distributed system, and, like the local ones, are not managed directly by it,
- explicit ones which are restricted to duplication of resources, that is to say to copy a same resource on different sites; such constraints are known and managed by the distributed system.

Figure 1 illustrates a distributed database with the different kinds of integrity constraints.

CALIF-CAR(license,make,vehicle-id,..)
CALIF-CAR-SPEC(make,model,power...)

The California local data-base schema

MASS-CAR(license,make,vehicle-id...)
MASS-CAR-SPEC(make,model,power...)

The Massachusetts local data-base schema

FRENCH-CAR(numero,construct.,vehicule-id...)
FRENCH-CAR-SPEC(marque,modele,puissance...)
FDRIVER(#ss,nom,prenom,age...)

The France local data-base schema

Figure 1 : An example of a distributed database

- Each local relation may be a resource.
- An implicit local integrity constraint may be:
  " any value of make in CALIF-CAR is a value of make in CALIF-CAR-SPEC "
- An explicit distributed integrity constraint may be:
  " CALIF-CAR-SPEC and MASS-CAR-SPEC must be identical "

Each transaction is supposed to be well specified, that is to say that the corresponding program respects the implicit integrity constraints: in fact, a transaction can be seen as the unit of consistent processing on a materialization (ROTH77) of a distributed database which can be simply defined as the set of resources manipulated by a transaction without the other copies.

The distributed database management system is responsible of many tasks, in particular :

(1) to ensure that each materialization given to each transaction is consistent; that means that all the implicit integrity constraints of that materialization must be verified (ESWA76);

(2) to ensure the capability of stoping a transaction before its completion and to restore the resources modified by the transaction in the state before modification (VERH78);

(3) to ensure the convergence of the databases, that is to say if the generation of transactions were stopped, all the resources which are duplicated would reach the same state; moreover, this state must generally correspond to a serializable state which would be obtain by a sequential execution of the transactions (BERN77);

(4) to ensure the possibility of operation when several sites are inoperational (ROTH77).

In order to perform the preceding tasks, the distributed system must contol the amount of concurrency. For this purpose, each site is decorated with a local controller. When a transaction wants to perform an operation on a resource , it submits the request to its own local controller : in the following, we call this controller the original controller. Generally, it has to establish communications with some remote contollers (see figure 2).

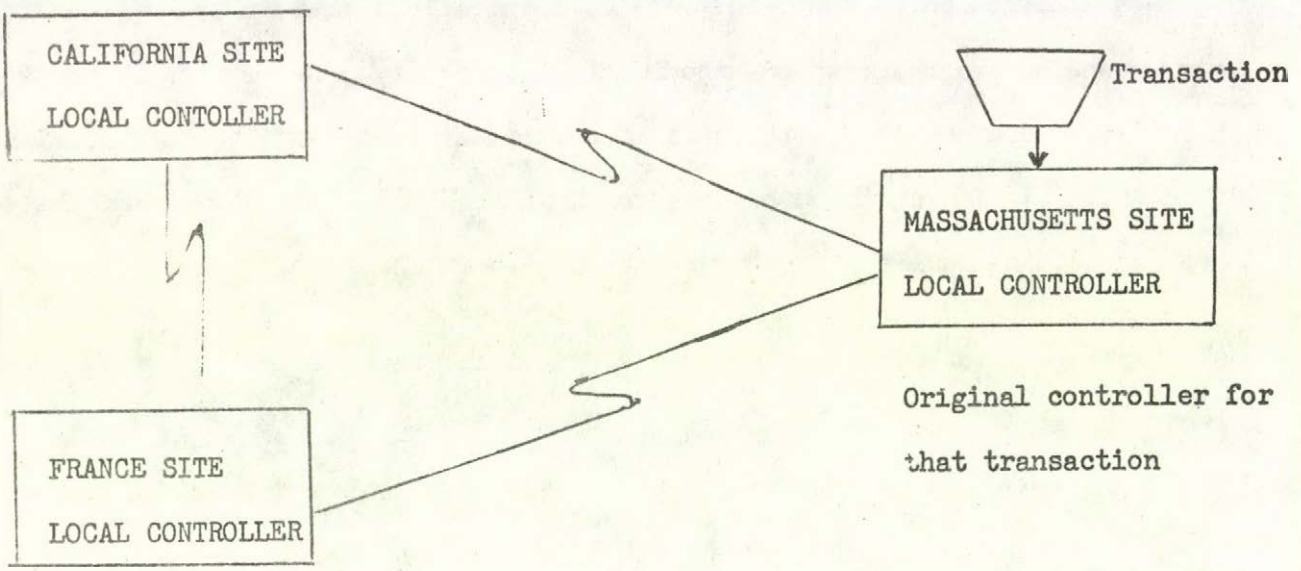


Figure 2 : a set of controllers

In order to ensure the possibility of operation when sites are inoperational, each controller must maintain a list of operational sites which is called the up-list (MENA78). Some algorithms need identical up-list over all the sites; that leads to synchronize the up-list when a site is detected as inoperational ; it may be impossible if the network can be partitionned.

In order to ensure convergence, it has been proved (ESWA76,MINO78) that transactions may be divided in two phases : a growing one where they really or virtually (BERN78) lock resources and a shrinking one where they unlock them. In the following, all the transactions are supposed to be two phases.

Moreover, in order to ensure the capability of restarting a transaction which has been stopped before its completion, a two steps commitment protocol (LAMP77) must be introduced: it permits to record the updatings of resources in the distributed database only at the end of each transaction; it is called the commitment point of the transaction. The two step feature allows to perform the commitment simultaneously on all the sites where the modified resources are located. The recovery procedure requires that after the updating of a resource on one site and

before the commitment of the transaction, it is forbidden another transaction to perform a new updating, unless the system cancels the first one. All the algorithms presented in the following respect this feature.

Finally, in order of simplicity, it is supposed that no queues are managed by the controllers: when a request cannot be performed, a reject response is given back to the transaction.

III. THE CENTRALIZED LOCKING ALGORITHM (ELLI77, MENA78)

The simplest method to avoid interference is to lock each resource being read or written by active transactions by the way of a central contoller, which is generally a chosen local controller but which can be a specific communication controller (HOLL77). This central controller is specialized to perform locking and unlocking of resource.

The centralized locking algorithm proceeds as follow:

(a) before performing an operation on a resource, a transaction requests for locking the resource in the desired operation modes (GARD76) to its original controller;

(b) one message is then sent by the original controller to the central controller requesting for the locking of all the copies of the resource;

(c) the central controller checks the requested lock; if it can be granted, then a grant message is sent back to the original controller; if not, a reject message is sent back to the original controller which returns a reject response to the transaction;

(d) once the original controller has received the grant message, the operation can be performed; if it is an updating operation, it is performed by sending simultaneously the result of the updating operation to all the sites which manage a copy and by waiting for an acknowledgment message from each of them;

(e) at the end of the transaction, after the two step commitment, all the resources used by the transaction are unlocked.

The main disadvantage of this solution appears with the inoperability of the central controller: it causes the inoperability of the whole distributed database system. In (MENA78) a solution has been proposed to this problem. However, it requests that a copy of the global lock table be managed by each site. Then, it is possible to change the central controller when a failure of the previous one has been detected; but, it has the serious disadvantage of multiplying the number of transmitted messages.

IV. THE DECENTRALIZED LOCKING ALGORITHM (ELLI77, STON78)

The decentralized solution partially presented in (ELLI77) and (STON78) avoids the failure problem and is able to continue updatings the database when one or more sites are inoperative. It proceeds as follow :

(a) as in step(a) of the previous algorithm, before performing an operation on a resource, a transaction requests for locking the resource to its original controller;

(b) one message is then sent by the original controller to each controller which manages a copy of the requested resource; simultaneously, a time-out is unlatched superior to the longest reasonnable transmission plus response time;

(c) when receiving an external request, each controller checks the requested lock; if it can be granted, then a grant message is sent back to the original controller; if not, a reject message is sent back to the original controller;

(d) when the original contoller receive a reject message,it must sent a cancellation of all the locking requests it has sent in step (b) even if they have been granted, and return a reject response to the transaction;

(e) when the original controller receives a grant message or detects a time-out, it must check if each invoqued site has granted the lock request or is inoperative; if yes and if at least one site is operative, the operation can be performed; if it is an updating operation, it is performed by sending simultaneously the result of the updating operation to all the sites which manage a copy and by waiting for an acknowledgment or a time-out response;

(f) at the end of the transaction, after the two step commitment, all the resources locked by the transaction must be unlocked;

(g) a special procedure is needed for recovering a site which has been repaired; such a procedure needs the history of the database; it can be performed by the way of asking it to any operational site (JOUV77) or by the persistent communication feature (SUTH74).Thus, the philosophy of ignoring sites which are inoperative leads to robust operation (ELLI77).

V. THE DAISY-CHAIN LOCKING ALGORITHM (CHU74)

This algorithm can be seen as a variant of the previous one. The daisy-chain is basically a cicular up-list, that is to say a chain of operative sites. It has been introduced the first time in order to prevent deadlock (CHU74) and can be used to prevent interferences. The algorithm must proceed has follow :

(a) as in step (a) of the previous algorithm, a transaction requests for locking a resource to its original controller;

(b) one message is then sent by the original controller to the first controller which follows it on the daisy chain and manage a copy of the requested resource;

(c) when a controller receive an external request, it checks the requested lock; if it can be granted, then the request is sent to the next controller on the dai chain; if not, a reject message is sent back to the previous controller via the daisy chain;

(d) when the original controller receives its own request, the operation can be performed as described in steps (e) and (f) of the previous algorithm;

(e) when a controller receive a reject message, it must release the correspondir lock if it is not the original contoller;

(f) when the original controller receives a reject, it must return it to the transaction as a response code;

(g) the daisy chain must be maintained when a site fails; for this purpose, when a controller sends a request, it must unlatch a time out and wait for a response message from its neighbour; when the time-out is completed, the request must be transmitted to the next site on the daisy-chain in a similar way ; it is importa to note that each site has only to know its two neighbour and not the whole dais chain;

(h) the recovery procedure when a site has been repaired is identical as in the previous algorithm; however,at the end of it, the site must call its two neighbo in order to reestabish it in the daisy-chain.

VI. THE PRIMARY SITE METHODOLOGY (ALSB76, STON78)

Between a centralized and a decentralized control, a method have been proposed which requires that all update activity for a given resource be funneled through a set of computers called the primary sites. In (STON78), two implementations of this methodology has been introduced. We present here an algorithm derived from the second one. For simplicity, a up-list of the whole network is assumed to be maintained on each site and no care is taken with network partitions (ROTH77).

The algorithm proceeds as follow :

(a) when a transaction submits a request for performing an operation, its original controller examines the up-list and calculate the k primary sites for the resource; they are the k first sites in the up-list which manage a copy of that resource;

(b) one message is then sent by the original controller to each primary site controller; this message requests the locking of the copies of the resource;

(c) each primary controller checks the requested lock; if it can be granted, then a grant message is sent back to the original controller; if not, a reject message is sent back to the original controller which return it to the transaction;

(d) once the original controller has received the k grant messages, the operation can be performed; if it is an updating operation, all the copies of the resource must be updated; it is performed as in the centralized algorithm;

(e) a reconfiguration procedure is needed to maintain the up-list; execution of such a procedure is performed as soon as a site fails; it can be performed as in the daisy chain algorithm; moreover, each operative site must known the new up-list in order to maintain a unique set of primarysites;

(f) the recovery procedure when a site has been repaired is identical to that of the centralized algorithm with in addition the inclusion of the repaired site in the up-list.

Let us point out that, by varying k from 1 to the number of operational sites, this locking algorithm goes from the centralized one with a different site controlling each resource to the decentralized one.

VII. THE DECENTRALIZED VOTING ALGORITHM (THOM76)

Another solution to the interference problem is a distributed voting algorithm suggested by THOMAS. Like in (GARC78), we only consider the daisy chain version of the algorithm. Such an algorithm is based on the association of a time stamp to each resource: this time stamp reflects the time at which the resource was assigned its present value.

The algorithm proceeds as follow :

(a) when a transaction submits an update request to its original controller, it includes as part of the request :

- a unique priority number as the request priority together with the site number,
- the name of the resource to be updated and its new value,
- a list of the resources upon which the update is based, called the base resources,
- the list of time stamps corresponding to the base resource values used to perform the updating;

this request is then passed around the daisy chain of all active sites;

(b) a controller, when it receives the request must perform one of the following action :

- vote reject if one or more of the base resouce time stamp is obsolete,
- vote accept only if the base variable time stamps are current and the request does not conflict with another current request; it conflicts if one active requet (one which is not accepted and not rejected) is trying to modify one base resource or if the resource to be updated is a base resource for another active request,
- vote deadlock reject if the base resources are current but the request conflicts with an active request of higher priority,
- defer voting if the base resources are current but the request conflicts with an active request of lower priority, or if the base resource time stamps of the request are more current than the corresponding local resources time stamps;

(c) when a controller notes that a majority consensus has been reached on a request it must perform the request and notify all other controllers to accept and perform

it ; moreover, the different active requests must be examined at each site, in order to reject those which are conflicting with the accepted request; the original controller must return an acceptation to the transaction as soon as it has been informed of it;

(d) when a controller reject a request, it must notify each other controller that the request has been rejected; the transaction must be informed of this rejection; moreover, when a controller is informed of a rejection, it must examine again deferred request and vote on it if it is possible;

(e) the recovery procedure when a site has been repaired is the same as in the decentralized locking algorithm; it is important to note that the number of sites considered in the voting procedure includes the inoperative one. Thus, the resynchronisation after repairing is facilitated.

Let us point out that the algorithm considers only updating requests: another algorithm is needed to insure consistency for readers. Such an algorithm can be find in (GARD78).

VIII. THE DECENTRALIZED TRANSACTION ORDERING ALGORITHMS (ROSE78,GARD78)

The previously presented algorithms were based on a locking schema. However, the voting algorithm is based on a detection of obsolete time stamps. The following algorithms are also based on a detection of illegal ordering of accesses to resources. These algorithms have been described for databases which are not duplicated, but they can be used in such an environnement. However, they need virtual locking (BERN78) of all the available copies used by a transaction. They are based on a detection of interferences with restart of one transaction when an interference occurs. They can be used as centralized or decentralized algorithm.

In this section, we present our algorithm with decentralized control . Then, the (ROSE78) algorithms are discussed as variants of this algorithm which proceeds as follow :

(a) when a transaction is initiated, the original controller gives it a serial identification number unique in the network; simple technique to do that have been described in (LELA78) and (LAMP78); then, the original controller send an initial message including this identification number to all the active sites where the transaction needs to operate;

(b) each controller manages a table of the transactions which are initiated on it; when it receives the initiating message, it determines the lowest local serial number of the initiated transactions; then, it returns back this number to the original controller;

(c) the original controller determines the minimum of all the local numbers; this number is associated to the transaction and will be refered as the stable number;

(d) each resource is labelled with the serial number of the last transaction which has updated it;

(e) when a transaction tries to read or update a resource, the access is(see figure 3 a) :

- accepted if the resource label is inferior to the transaction stable number or equal to the transaction serial number (access after updating),
- rejected in all the other cases; the transaction must then be roll-backed;

(f) each concerned controller must be informed of the end of a transaction; the commitment of a transaction is locally performed by extracting its serial number from the table of the initiated transaction;

(g) when a site is inoperative, of course all the transactions which are initiated on it must be rol-backed; then the site can be ignored; when it has been repaired the recovery procedure must perform the accepted updatings and find a correct table of the transactions initiated on it by asking to all the sites which manage a copy of a part of its local database.

The (ROSE78) DIE-WAIT schema can be presented as a variant of the previous algorithm where step (e) is replaced by :

(e') when a transaction tries to read or update a resource, the access is (see figure 3b) :

- accepted if the resource label is inferior to its stable number or equal to its serial number,
- delayed if the resource label is included between its stable number and its serial number,
- rejected in all the other cases; the transaction must then be roll-backed.

The (ROSE78) WOUND-WAIT schema can be presented as another variant where step (e) is replaced by:

(e") when a transaction tries to read or update a resource, the access is (see figure 3c):

- accepted if the resource label is inferior to its stable number or equal to its serial number,
- delayed if the resource label is included between its stable number and its serial number,

- accepted in all other cases but the transaction the serial number of which is the resource label must be roll-backed.

Both the (ROSE78) algorithms decrease the risk of roll-backs but need the management of waiting queues.

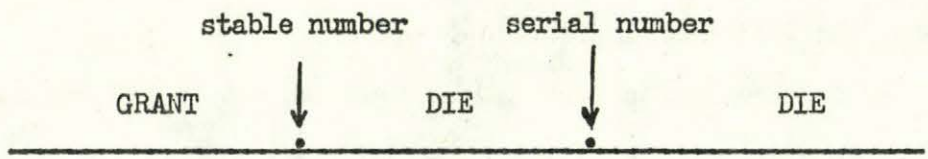


Figure 3a

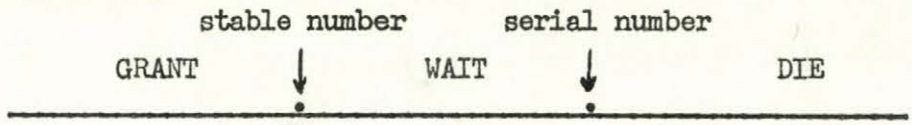


Figure 3b

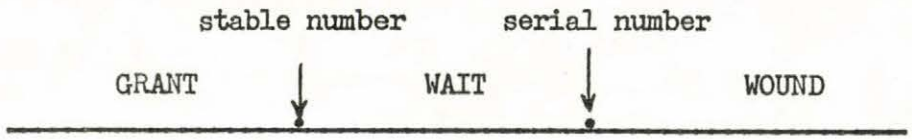


Figure 3c

Figure 3 : Illustration of the transaction ordering algorithms

IX. THE PREANALYSIS METHODOLOGY (BERN77, BERN78)

The previously presented algorithms do not use an apriorist knowledge of the transaction behaviour. The SDD.1 updating methodology is quite different. Interferences resolution is performed by the way of preanalysing the whole set of transactions with the help of a conflict graph. When the transaction programs are defined, the sets of resources they are able to read or write are specified. Then, a conflict graph is constructed as follow : each node correspond to a read or write set of resources and an undirected edge links two nodes belonging to a same transaction or two nodes representing non-disjoint sets of resources.

Four different synchronisation protocols are available. By the way of the conflict graph, transactions are classed in one of four classes, each of them corresponding to one level of protocols. The most efficient protocol specifies no inter-computer synchronisation at all; a transaction which runs under this class needs only perform local locking in order to ensure the convergence of the distributed database. This protocol can be used when no risk of interferences is detected with the help of the conflict graph. Each of the other protocol introduce some degree of non-local synchronisation. The less efficient protocol requires global locking of all the resources.

When a transaction is activated, a protocol selection function operates by mapping each transaction entered into a class and by selecting the adequate protocol. This function uses tables predefined by the database administrator. The ultimate goal of this strategy is to reduce the number of messages interchanged to synchronise the transactions and to avoid global locking when it is possible.

X. CONCLUSION

Several algorithms have been described which can be used for updating redundant distributed databases. The choice of one is not obvious. Many criteria must be considered as :

(a) the degree of parallelism allowed in the distributed system;

(b) the completness of the proposed solution to the interference problem and to the subsequent one which are deadlock (GOLD77, MUNT78) and permanent blocking or cyclic restart (HOLT71, ROSE78);

(c) the response time given to a transaction which performs an operation on a resource; it can vary from only a local computation to several messages exchanged on the network;

(d) the promptness for updating the copies of a resouce (GELE78);

(e) the cost of the message traffic generated (GARC78);

(f) the tolerance to different kinds of failures and the simplicity of restating after reparation;

(g) the degree of prevision required to the transactions; some algorithms need that each transaction forsee all the accessed resource before starting; others need to lock the resource before each operation; others have no such requirements;

(h) the cost in I/O time, processing time and memory requirement (for the lock tables, the time stamps...).

All these criteria may be important. The opinion of the author is that the weight of each is dependent upon the considered application. Consequently, each futur distributed database management system should be able to propose several methodologies; thus, the database administrator would be able to choose the best one for each application.

REFERENCES

(ADIB78) M. ADIBA,J.C. CHUPIN,R. DEMOLOMBE,G. GARDARIN,J. LEBIHAN
"A technical overview of resarch and development in distributed systems"
4th international conference on Very Large Database BERLIN Sept 1978

(ALSB76) P. A. ALSBERG,G.G. BELFORD,S.R. BRUNCH
"Synchronization and Deadlock"
CAC Document Number 185- University of Illinois- March 1976

(BACH77) C. W. BACHMAN
"Advances in Database Technology"
Infotech State of the Art Tutorial - LONDON December 1977

(BERN77) P.A. BERNSTEIN,N. GOODMAN,J.B. ROTHNIE,C.A. PAPADIMITRIOU
"Analysis of serializability in SDD.1: a system for distributed databases"
Computer Corporation of America - Technical report N CCA-77-05

(BERN78) P.A. BERNSTEIN, D,W. SHIPMAN
"A formal model of concurrency control mechanisms for database systems"
3rd Berkeley Workshop on Distributed Data Management and Computer Networks
Pp 189-205 August 78 - BERKELEY

(CHU74) W.W. CHU,G. OHLMACHER
"Avoiding deadlock in distributed databases"
ACM National Conference - Pp 156-160 - Nov. 1974

(DAVE78) R.A. DAVENPORT
"Integrity in distributed database systems"
Eurocomp 78 Pp 751-773

(ELLI77) C.A. ELLIS
"A robust algorithm for updating duplicate databases"
2nd Berkeley Workshop on Distributed Data Management and Computer Networks
Pp 146-158 May 1977 - BERKELEY

(ENGL76) R.W. ENGLES

"Currency and concurrency in the cobol database facility"

Modelling in DBMS-- North Holland Pub Co. Ed. by J. NIJSSEN Pp624-633 1976

(ESWA76) K.P. ESWARAN,J.N. GRAY,R.A. LORIE,I.L. TRAIGER

"The notion of consistency and predicate locks in a database system"

Comm. ACM - Vol19/11 - Nov 1976 Pp 624-633

(FRY 75) J.P. FRY,M.E. DEPPE

"Distributed database : a summary of research"

Techmical Report 75DE-2 University of MICHIGHAN- ANN ARBOR 1975

(GARC78) H. GARCIA MOLINA

"Performance comparaison of two update algorithms for distributed databases"

3rd Berkeley Workshop on Distributed Data Management and Computer Networks

Pp 108-122 August 1978 BERKELEY

(GARD76) G. GARDARIN,S. SPACCAPIETRA

"Integrity of databases:a general lockout algorithm with deadlock avoidance"

Modelling in DBMS - North Holland Pub Co. Ed. by J. NIJSSEN Pp395-411 1976

(GARD77) G. GARDARIN,P. LEBEUX

"Scheduling algorithms for avoiding inconsistency in large databases"

3rd international conference on Very Large Databases TOKYO oct 1977

(GARD78) G. GARDARIN

"Resolution des conflits d'acces simultanes a un ensemble d'informations

Applications aux bases de donnees reparties"

These d'Etat Universite de PARIS VI Avril 1978

(GELE78) E. GELENBE,K. SEVCIK

"Analysis of update synchronisation for multiple copy databases"

3rd Berkeley Workshop on Distributed Data Management and Computer Networks

Pp 69-90 August 1978 BERKELEY

(GOLD77) B. GOLDMAN

"Deadlock detection in computer networks"

Thesis Mass. Institute of Technology June 1977

(HOLL77) E. HOLLER, O. DROBNIK

"Implementation of decentralized coordination mechanisms in distributed mini-micro computer systems"

Technical Report-Institut fur Datenverarbeitung in der Technik

KARLSRUHE RFA 1977

(HOLT71) R.C. HOLT

"Comment on prevention of system deadlocks"

Comm. ACM - Vol 14/1 - Jan 1971 Pp 36-38

(JOUV77) M. JOUVE

"Reliability aspects in a distributed database management system"

AICA'77 Proc. Data Bases Pp 199-209

(LAMP77) B. LAMPSON, H. STURGIS

"Crash recovery in a distributed data storage system"

Internal report, Computer Science Laboratory

XEROX PALO ALTO Research Center - 1976

(LAMP78) L. LAMPORT

"Time, clocks and the Ordering of Events in a Distributed System"

Comm. ACM July 1978 V21/7 Pp 558-565

(LELA78) G. LE LANN

"Algorithms for distributed data-sharing which use tickets"

3rd Berkeley Workshop on Distributed Data Management and Computer Networks

Pp 259-272 August 1978 BERKELEY

(MENA78) D.A. MENASCE, G.J. POPEK, R.R. MUNTZ

"A locking protocol for resource coordination in distributed databases"

SIGMOD Austin TEXAS to be published in TODS

(MUNT78) D.A. MENASCE, R.R. MUNTZ

"Locking and deadlock detection in distributed databases

3rd Berkeley Workshop on Distributed Data Management and Computer Networks

Pp 215-234 August 1978 BERKELEY

(MINO78) T. MINOURA

"Maximally Concurrent Transaction Processing"

3rd Berkeley Workshop on Distributed Data Management and Computer Networks

Pp 206-214 August 1978 BERKELEY

(ROSE78) D.J. ROSENKRANTZ, R.E. STEARNS, P.M. LEWIS

"System level concurrency control for distributed database systems"

ACM TODS Vol 3/N3 June 1978 Pp 178-198

(ROTH77) J.B. ROTHNIE, N. GOODMAN

"A survey of research and development in distributed database management"

3rd international Conference on Very Large Databases TOKYO oct 1977

(STON78) M. STONEBRAKER

"Concurrency control and consistency of multiple copies of data in distributed INGRES"

3rd Berkeley Workshop on Distributed Data Management and Computer Network

Pp 235-258 August 1978 BERKELEY

(SUTH74) W.R. SUTHERLAND

"Distributed Computation Research at BBN"

Vol 111 - BBN Technical Report N 2976 - Dec 1974

(THOM76) R.H. THOMAS

"A solution to the update problem for multiple copy database which use distributed control"

BBN Report N 3340 July 1975

(TRIN75) M. TRINCHIERI

"On managing interference caused by database sharing"

ALTA FREQUENZA 1975 N11 Vol XLIV Pp 641-650

(VERH78) J.S. VERHOFSTAD

"Recovery techniques for database systems"

Computing Surveys, Vol 10,N2,June1978

The asymptotic distribution of the cost function of an adaptive system recovery procedure

A. Krámli - P. Lukács

Computer and Automation Institute Hungarian Academy of Sciences

Our aim is to determine the asymptotic distribution of the cost function of an adaptive version of system recovery procedure. This adaptive system recovery procedure is described by Benczur and Krámli in [1] as suggested by a result of Gelenbe [4] .

The recovery procedure as a method for providing the integrity of data base systems generates control points (C.P.) (i.e. generates a copy of the data base) at times determined by a decision method. If a failure occurs the copy of the last control point will be reloaded and the processed tasks from the last control point will be repeated.

The failure process (F.P.) $\{\tau_j\}$ is an inhomogenious Poisson point process. Let $\{\nu_i\}$ be independent random variables geometrically distributed with parameter p and independent of the Poisson process. The inhomogenity of the Poisson process is gouverned by $\{\nu_i\}$ in the following way: let $\theta_k := \sum_{i=1}^{k} (\nu_i + 1)$ $k = 0,1,2,\dots$

$$P\left(\tau_{\theta_k + j} < x\right) = 1 - e^{-\lambda_1 x} \qquad j = 1,2,\dots,\nu_{k+1} \qquad k = 0,1,2,\dots$$

$$P\left(\tau_{\theta_k} < x\right) = 1 - e^{-\lambda_2 x} \qquad k = 1,2,3,\dots \quad \text{where } \lambda_2 < \lambda_1$$

The time intervals $\{(\tau_{\theta_k}, \tau_{\theta_{k+1}-1})\}_{k=0}^{\infty}$ are the so called heavy periods of the failure process, that will be an illustration of the proverb "trouble never comes alone". It is evident that the "heavy" and "light" periods form an alternate Poisson process with parameter $p\lambda_1$ and λ_2.

The decision method for the control point generation depends on three parameters t_1, t_2 and d. If the time since the last failure occured is greater then a certain d the system generates control points by equidistant steps t_2, in the other case by t_1 where $t_1 < t_2$.

The Semi-Markov process with finite state space describing the recovery procedure was constructed by Benczur and Krámli in the following way:

$$\eta(t) = \begin{cases} 1 \\ 2 \\ 3 \\ 4 \end{cases} \underline{\text{if}} \text{ the F.P.} \begin{cases} \text{is} \\ \text{is} \\ \text{isn't} \\ \text{isn't} \end{cases} \text{in heavy period } \underline{\text{and}} \begin{matrix} \text{the steps} \\ \text{between} \\ \text{CP-s have} \\ \text{length} \end{matrix} \begin{cases} t_1 \\ t_2 \\ t_1 \\ t_2 \end{cases}$$

It is convenient to suppose the trajectories of $\eta(t)$ to be left continous.

This process has an embedded ergodic Markov chain $\{\eta(n)\}$ with the following transition probability matrix P and unique stationary distribution π

$$P = \begin{pmatrix} 0 & \frac{e^{-\lambda_1 d}}{1-q(1-e^{-\lambda_1 d})} & 1-p_{12} & 0 \\ 1-p & 0 & p & \\ 1-e^{-\lambda_2 d} & 0 & 0 & e^{-\lambda_2 d} \\ 1 & 0 & 0 & 0 \end{pmatrix} \text{(see [1])},$$

where $P = (p_{ij})_{i,j=1}^{4}$.

$$\pi = \left(p+(1+q)e^{-\lambda_1 d},\ e^{-\lambda_1 d},\ p,\ e^{-e^{\lambda_2 d}}\cdot p\right)\frac{1}{(1+q)e^{-\lambda_1 d}+pe^{-\lambda_2 d}+2p}$$

For the cost calculation let us define the following secondary random variables for the Markov process as in [1] :

δ_n: = sojourn time
μ_n: = # of failures
τ_n: = extra run time
C_n: = # of loads and dumps

in the time interval $(t: \eta(t)=\eta_n)$

If the initial distribution for the Markov chain $\{\eta_n\}$ is its stationary one then the $\{(\delta_n, \mu_n, \tau_n, C_n)\}$ process is also stationary. Morover it fulfils the uniform strong mixing condition, i.e. if $\mathcal{A}_t^s$, $(t \le s)$ denotes the σ -algebra generated by the random variables $(\delta_m, \mu_m, \tau_m, C_m)$ $m = t, t+1, \dots, s$ then

$$\varphi(n) = \sup_{\substack{\tau,\ A\in\mathcal{A}_1^s \\ B\in\mathcal{A}_{s+n}^{\tau}}} \frac{P(AB)-P(A)\cdot P(B)}{P(A)} \longrightarrow 0.$$

This last statement is a consequence of a similar result for ergodic Markov chains with finite state space ([3] pp 167-168) and the conditional independence of the random variables $(\delta_n, \mu_n, \tau_n, C_n)$ if the trajectory of $\{\eta_n\}$ is fixed. We also know that the rate of convergence of $\varphi(n)$ is exponential, i.e. there are a positive constant A and $\rho<1$ such that $\varphi(n)<A\rho^n$. (*)

Let α_N be the first moment when the normal working time exceeds N

i.e. $\alpha_N = k$ if for $j<k$ $\sum_{n=1}^{j}(\delta_n - t_n) < N$ and $\sum_{n=1}^{k}(\delta_n - t_n) \geq N$

As a consequence of the strong low of large numbers and the conditional independence of the variables (δ_n, t_n) if the trojectory of $\{\eta_n\}$ is fixed α_N / N tends to $E^{-1}(\delta - t)$ in probability measure.

Let our cost function have the following form

$$f\left(\sum_{n=1}^{\alpha_N} \xi_n\right) \qquad \text{where } \xi_n = A\delta_n + B\mu_n + Ct_n + D\cdot c_n$$

and f is continous.

Let $\overline{\xi}_n$ be the centralized version of ξ_n. The observations made above mean that the sufficient conditions for the validity of central limit theorem for the sum $\alpha_N^{-1/2} \sum_{n=1}^{\alpha_N} \overline{\xi}_n$ are fulfilled

(see[3] Th 2o.3) :

(i) $\sum_{n=1}^{\infty} \varphi(n)^{1/2} < \infty$

(ii) $\alpha_N / a_N \xrightarrow{P} \theta$ where $a_N \to \infty$ α_N is positive integer valued random variable, θ is positiv random variable

(iii) $\sigma^2 = E(\xi_1 - E\xi_1)^2 + 2\sum_{k=2}^{\infty}(\xi_1 - E)(\xi_k - E) > 0.$

First (i) follows from (*)
The quantity σ^2 can be explicitly calculated and we see it is strictly positive.

So we have proved that

$$P\left(\frac{\sum_{n=1}^{\alpha_N}\bar{\xi}_n}{\sigma\sqrt{\alpha_N}} < x\right) \longrightarrow \Phi(x)$$

where Φ is the standard normal distribution function.
The quantity σ^2 can be explicitly calculated as

$$\sigma^2 = \lim_{n\to\infty}\frac{1}{n}E\left(\sum_{k=1}^{n}(\xi_k - E\xi_k)\right)^2 =$$

$$= \lim_{n\to\infty}E\left(\frac{1}{n}E\left(\left(\sum_{k=1}^{n}(\xi_k - E\xi_k)\right)^2 \middle| \eta_1,\dots,\eta_n\right)\right) =$$

$$= \lim_{n\to\infty}E\left(\frac{1}{n}\sum_{k=1}^{n}\left(E(\xi_k^2|\eta_{k-1},\eta_k) - E(\xi_k|\eta_{k-1},\eta_k)\right)\right) +$$

$$+ \lim_{n\to\infty}E\left(\frac{1}{n}\left(\sum_{k=1}^{n}E(\xi_k|\eta_{k-1},\eta_k) - E\xi_k\right)^2\right)$$

The first expression tends to $\sum_{i,j=1}^{4}p_i p_{ij}(M_{ij} - E_{ij})$
as a consequence of the strong low of large numbers and the ergodic theorem of Markov chains, where

$M_{ij} := E(\xi_k^2 \mid \eta_{k-1}=i, \eta_k=j)$ and $E_{ij} := E(\xi_k \mid \eta_{k-1}=i, \eta_k=j)$.

This limit quantity is strictly positive. The second expression is non-negative and equal to

$$E\left(\left(E(\xi_1|\eta_0,\eta_1)-E\xi_1\right)^2\right)+$$

$$+2\sum_{k=2}^{\infty}E\left(\left(E(\xi_1|\eta_0,\eta_1)-E\xi_1\right)\left(E(\xi_k|\eta_k,\eta_{k+1})-E\xi_k\right)\right)=$$

$$=\sum_{i,j=1}^{4}p_i p_{ij}\left(E_{ij}^2+E_{ij}E_j\right)-2E^2+$$

$$+2\sum_{i,j=1}^{4}p_i p_{ij}\left(E_{ij}-E\right)\cdot\sum_{m=1}^{4}\left(\sum_{k=3}^{\infty}\left(p_{jm}^{k-1}-p_m\right)\right)E_m$$

where $E_j:=\sum_{m=1}^{4}p_{jm}E_{jm}$ and $E:=\sum_{i,j=1}^{4}p_i p_{ij}E_{ij}$

As P is ergodic

$$\left(\sum_{k=3}^{\infty}\left(p_{jm}^{k-1}-p_m\right)\right)_{m,j=1}^{4,4}=\left(P^2-\Pi\right)\left(I-P+\Pi\right)^{-1}$$

where $\Pi=\begin{pmatrix}p_1 & p_2 & p_3 & p_4\\ p_1 & p_2 & p_3 & p_4\\ p_1 & p_2 & p_3 & p_4\\ p_1 & p_2 & p_3 & p_4\end{pmatrix}$

For the explicit calculation of σ^2 we need the first and second conditional moment of the secondary variables if the trajectory of $\{\eta_n\}$ is fixed.

For illustration we give these quantities only at the transition from $\eta_k=1$ to $\eta_{k+1}=2$.

The first conditional moments of secondary variables are determined in [1] , so we need to calculate their second moments only.

Let us introduce some general notations for this pourpose.

$$h_t(x) := x - nt \quad \text{if} \quad x \in [nt, (n+1)t) \quad n = 0,1,\ldots$$

$$E^{(\lambda)}_{t,d} := \int_0^t h_t(x)\, \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda d}}\, dx \qquad d \in (0\,\infty]$$

$$e^{(\lambda)}_{t,d} := \sum_{k=0}^{\infty} \int_{kt}^{\infty} \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda d}}\, dx$$

$$M^{(\lambda)}_{t,d} := \int_0^t h_t^2(x)\, \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda d}}\, dx$$

$$m^{(\lambda)}_{t,d} := \sum_{k=1}^{\infty} (2k-1) \int_{kt}^{\infty} \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda d}}\, dx + 1$$

As we suppose that the initial distribution of the Markov chain is its stationary one, we can omit the time index n from η_n and ξ_n.

Let $E_{12}\xi := E\left(\xi_n \,\middle|\, \eta_{n-1}=1,\ \eta_n=2\right) = E\left(\xi \,\middle|\, \eta_{left}=1,\ \eta_{right}=2\right)$

Let γ be the number of the failurless ones of the intervals between consecutive check points in the time intervals of the form $\{t: \eta(t)=1\}$. Recall that in such time interval the length of the time intervals between check points is t_1.

It is easy to see that the distribution of γ is the following:

$$P(\gamma=k)=\frac{1-e^{-\lambda_1 t_1}}{1-e^{-\lambda_1 d}}\,e^{-\lambda_1 t_1 k} \qquad k<\left[\frac{d}{t_1}\right],$$

$$P\left(\gamma=\left[\frac{d}{t_1}\right]\right)=\frac{1-e^{-\lambda_1 t_1\left\{\frac{d}{t_1}\right\}}}{1-e^{-\lambda_1 d}}\,e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}$$

where $[x]$ is the integer part of x and $\{x\}$ is its fraction part.

Let ϱ_m be defined as the extra run time caused by the m-th failure in the time interval $\{t: \eta(t)=1\}$ Of course the $\{\varrho_m\}$ are independent and identically distributed random variables with mean:

$$E_{12}\varrho=\frac{1}{\lambda_1}-t_1\frac{e^{-\lambda_1 t_1}}{1-e^{-\lambda_1 t_1}}\cdot\frac{1-e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}}{1-e^{-\lambda_1 d}}-t_1\left\{\frac{d}{t_1}\right\}\cdot\frac{e^{-\lambda_1 d}}{1-e^{-\lambda_1 d}}$$

and with second moment:

$$E_{12}\varrho^2=\frac{2}{\lambda_1^2}-\left(t_1^2+\frac{2t_1}{\lambda_1}\right)\frac{e^{-\lambda_1 t_1}}{1-e^{-\lambda_1 t_1}}\cdot\frac{1-e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}}{1-e^{-\lambda_1 d}}-$$

$$-\left(t_1^2\left\{\frac{d}{t_1}\right\}^2-\frac{2t_1\left\{\frac{d}{t_1}\right\}}{\lambda_1}\right)\frac{e^{-\lambda_1 d}}{1-e^{-\lambda_1 d}}$$

The covariance between ϱ and γ is

$$E_{12}\varrho\gamma = \frac{1}{1-e^{-\lambda_1 d}}\left(\left(\frac{1}{\lambda_1}-t_1\frac{e^{-\lambda_1 t_1}}{1-e^{-\lambda_1 t_1}}\right)\cdot\left(e^{-\lambda_1 t_1}\frac{1-e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}}{1-e^{-\lambda_1 t_1}}-\left[\frac{d}{t_1}\right]e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}\right)+\right.$$

$$\left.+\left(\frac{1}{\lambda_1}-t_1\left\{\frac{d}{t_1}\right\}\frac{e^{-\lambda_1 t_1\left\{\frac{d}{t_1}\right\}}}{1-e^{-\lambda_1 t_1\left\{\frac{d}{t_1}\right\}}}\right)\cdot\left(1-e^{-\lambda_1 t_1\left\{\frac{d}{t_1}\right\}}\right)\left[\frac{d}{t_1}\right]e^{-\lambda_1 t_1\left[\frac{d}{t_1}\right]}\right).$$

$$E_{12}\delta^2 = E_{12}\mu\cdot M^{(\lambda_1)}_{d,d}+\left(E_{12}\mu^2-E_{12}\mu\right)\left(E^{(\lambda_1)}_{d,d}\right)^2+2dE_{12}\mu\cdot E^{(\lambda_1)}_{d,d}+d^2$$

$$E_{12}\tau^2 = E_{12}\mu\cdot M^{(\lambda_1)}_{t_1,d}+\left(E_{12}\mu^2-E_{12}\mu\right)\left(E^{(\lambda_1)}_{t_1,d}\right)^2$$

$$E_{12}\mu^2 = \frac{1+q(1-e^{-\lambda_1 d})}{(1-q(1-e^{-\lambda_1 d}))^2} \quad \text{because} \quad P_{12}(\mu=k)=\left(1-q(1-e^{-\lambda_1 d})\right)q^k(1-e^{-\lambda_1 d})^k$$

$$E_{12}c^2 = E_{12}\mu\cdot m^{(\lambda_1)}_{t_1,d}+\left(E_{12}\mu^2-E_{12}\mu\right)\left(e^{(\lambda_1)}_{t_1,d}\right)^2$$

$$E_{12}\delta\tau = dE_{12}\tau+E_{12}\mu\cdot\left(E_{12}\varrho+t_1\cdot E_{12}\varrho\gamma-E^2_{12}\varrho-t_1E_{12}\varrho\cdot E_{12}\gamma\right)-$$

$$-E_{12}\mu^2\left(E_{12}\varrho+t_1E_{12}\varrho\cdot E_{12}\gamma\right)$$

$$E_{12}\tau\mu = E_{12}\mu^2\cdot E_{12}\varrho$$

$$E_{12}\delta\mu = E_{12}\tau\mu+t_1E_{12}\mu^2\cdot E_{12}\gamma+d\cdot E_{12}\mu$$

$$E_{12}\tau\cdot c = E_{12}\mu^2\cdot E_{12}\rho(1+E_{12}\gamma) + E_{12}\mu\left(E_{12}\rho\cdot\gamma - E_{12}\rho\cdot E_{12}\gamma\right)$$

$$E_{12}\delta\cdot c = d\cdot E_{12}c + E_{12}\mu\left(E_{12}\rho\cdot\gamma + t_1 D_{12}^2\gamma\right) +$$

$$+ E_{12}\mu^2\left(E_{12}\rho + t_1\left(E_{12}^2\gamma + E_{12}\gamma\right)\right)$$

$$E_{12}\mu\cdot c = E_{12}\mu^2\left(E_{12}\gamma + 1\right).$$

References

[1] A. Benczur - A. Krámli: On Integrity of Data Base Conference preprints of the 4 th International Symposium on Modelling Performance Evaluation of Computer Systems 1979. Vol 2.

[2] A. Benczur - A. Krámli - J. Pergel: A central limit theorem for the extra run time caused by failures in a data base management system /to appear/

[3] Billingsley: Convergence of Probability Measures. New York - London ... Wiley 1968.

[4] Erol Gelenbe: On the optimum checkpoint interval /private communication/ 1977.

QUEUEING SYSTEMS WITH WAITING TIME DEPENDENT SERVICE TIMES

/Analysis and optimization/

A.Wolisz, J.Izydorczyk

1. Introduction

Recently an increasing interest can be observed in system performance evaluation, equally for computer systems, production systems, transportation systems etc. Among tools utilized in system performance evaluation, the queueing theory approach can be pointed out as one of the most powerful and therefore this area of science is being extensively developed.
A lot of new queueing models reflecting special features of investigated real - life systems have been studied. For example investigations in the area of computer systems performance evaluation gave reason for a rapid developement of priority queueing models and time-sharing models. This application, as well as industrial engineering and computer networks, stimulated also a big affort in the area of multi - stage queueing systems and networks of queues.

Among the relatively new, nonclassic service systems one can distinguish an interesting group of queueing models, with service time of every customer being dependent on waiting time spent in the queue by this very customer. Such an phenomenon occurs in numerous real-life situations.

An example which animated the majority of investigations in this area is the material flow between steel-plant and blooming mill in metalurgical industry.

Obtained in the LD- conversion process steel is poured into special moulds and left for some predetermined time period justified by metalurgical reasons in order to assure proper crystallization. After this period moulds are removed in a stripping bay, and ingots are transported to the soaking pits department in order to be heated. The heating should assure an / theoretically/ constant temperature in every point of the ingot, slightly lower than the melting temperature. This is necessary for assuring proper rolling conditions.

The stream of ingots as well as the heating duration has a stochastic nature, thus a queue of ingots waiting for charging into the pits forms in front of the soaking pits division. Waiting ingots are subject to extensive heat losses and the heating duration is evidently dependent on waiting time.

As an example of such a dependence in the area of computer system performance evaluation one can point out (cf. [LIBU 74]) flight control systems. In the case when a request for trajectory correction is waiting in the queue of jobs /perhaps identical in nature but issued by other planes/ it's service time may be increased, as a need for either more sophisticated position calculations or additional radar measurements will probably occure.

In this paper we shall present state of the art in the area of queueing systems with waiting time dependent service times. Additionally we shall give some examples illustrating special features of such systems. The error which would occure had the dependence of service time upon waiting time been neglected will also be investigated.

2. Basic definitions

In further considerations we shall deal with service systems having following features:

a/ Demands arrive one by one in time epochs $t_1, t_2, \ldots, t_n, \ldots$ and time periods $a_n = t_{n+1} - t_n$ are independent, identically distributed*) random variables with distribution function $A(x)$, thus the input process is a renewal process.

b/ Demands are served individually, service time b_w of every demand is dependent on it's waiting time w and given by a conditional distribution function $B(x/w)$.

We shall introduce for such systems a modified Kendall notation

$$\alpha/B_w \;/\; n \;/\; N \;/\; K \;/\; DISC, \qquad (2.1)$$

where $\alpha, n, N, K, DISC$ have the classical meaning, [KLEI 75] and stand respectively for the shape of distribution $A(x)$, number of parallel service stations, dimension of the source, the maximal number of demands accepted by the system and the service discipline, while B_w states, that the system has service time dependent upon waiting time, this dependence being given by an additionally defined conditional distribution function $B(x/w)$.

Further we shall be interested in the throughput of the considered systems. Let us define the maximal throughput T_{DISC} of a given system under the scheduling discipline DISC as a supremum of the demand's stream intensity λ, for which the considered system is ergodic.

---

*) In this paper the following notation will be used:
Let r be a random variable. The probability density function (p.d.f) of this variable will be denoted $r(x)$, distribution function $R(x)$ and the Laplace - Stieltjes transform of this distribution function $\bar{r}(s)$, $\bar{r}(s) = \int_0^\infty e^{-st} \, d\,R(t)$.
Mean value of the variable r will be denoted $E(r)$.

It is worth mentioning that for classical systems with $B(x/w) = B(x)$, T_{DISC} is independent of queueing discipline (cf [GROS 74]) and equal to

$$T = \frac{1}{E(b)} \quad . \tag{2.2}$$

if only server is not idle as long as there are jobs in the queue.

3. Steady state conditions under FIFO discipline .

We shall start our survey with a discussion of steady state conditions for the FIFO queueing discipline, while all so far obtained results for other queueing disciplines will be presented in section 6.

The first results in this area have been reported in [SUGA 65] where a system $GI/B_w/n$ with

$$B(x/w) = \eta(x - \varphi(w)) \quad , \quad \eta(y) = \begin{cases} 0 & y \leq 0 \\ 1 & y > 0 \end{cases}$$

was investigated. In this case $b = \varphi(w)$ describes the deterministic dependence of service time upon waiting time.

The function $\varphi(w)$ has following features:

$$\left.\begin{aligned} &\varphi(0) > 0 \\ &\lim_{w \to \infty} \varphi(w) = \varphi(\infty) < \infty \end{aligned}\right\} . \tag{3.1}$$

It has also been assumed, that the mean interarrival time $E(a)$ is finite, and demands are served on n identical, independently operating servers.

It has been proved that the above defined system is ergodic if

$$n \cdot E(a) > \varphi(\infty) . \tag{3.2}$$

On the other hand, if

$$Pr\{n \cdot a < \varphi(0)\} = 0 , \tag{3.3}$$

then the queue remains empty even if $n \cdot E(a) < \varphi(\infty)$.

A more general dependence of service time upon waiting time was considered in [CALL 73].

A class of discrete systems was investigeted with variables a, w, b_w, standing for interarrival time, waiting time and service time being discrete, with integer, nonnegative values. Additionally it was assumed that

$$Pr\{b_w - a = j\} > 0 \quad \text{for} \quad j = -1, 0, 1; \quad w = 0, 1 \ldots$$

It has been proved, that the considered system is ergodic if

a/ $E(b_w) < \infty \quad \text{for} \quad w = 0, 1, 2 \ldots$ (3.4)

b/ $\limsup_{w \to \infty} E(b_w) < E(a)$. (3.5)

The author conjectured that a similar condition holds also for the continous case, which has been later investigated in [TWEE 75]. Let us define a random variable $z_w = b_w - a$, having a p.d.f. $z_w(x)$. We shall assume that $z_w(x) > 0$ for all $x \in (-\delta, \delta)$ where δ is a positive constant independent of w. Under these assumptions it has been proved that the considered system is ergodic, if

(i) $b_w(x)$ is continous in w for every fixed x,

(ii) there exist positive constants N, B, ε such that

$$E(b_w) \leqslant B < \infty, \quad w \leqslant N;$$

$$E(b_w) \leqslant E(a) - \varepsilon, \quad w > N.$$

Condition (ii) can be rewritten in the form

(ii)' the mean service time $E(b_w)$ is a continous function in w and is bounded for every w,

(ii)'' $\limsup_{w \to \infty} E(b_w) < E(a)$

which together with (i) give an analogy to conditions (3.4) and (3.5).

4. Steady - state analysis for FIFO discipline.

In this section we shall present a methodology*) for analysis of open, single server queueing systems with waiting time dependent service times in the case of FIFO discipline. This description will be followed by a list of special cases which have so far been solved in detailes.

Let us investigate two consecutively served demands, "n" and "n+1" having waiting times w^n , w^{n+1} and service times b_w^n , b_w^{n+1} respectively. According to earlier assumptions

$$\underset{n\in\{1,2,\dots\}}{\forall} \quad B(x/w) = Pr\{b_w^n \leq x\}. \tag{4.1}$$

Waiting time w^{n+1} can be derived as

$$w^{n+1} = \max[0, w^n + b_w^n - a^{n+1}]. \tag{4.2}$$

For a stochastic process $\{w^n\}$, n=1,2,..., defined by (4.2) the distribution function of (n+1) -th demand waiting time $W^{n+1}(x)$ can be expressed in terms of $W^n(x)$ as

$$W^{n+1}(x) = \int_{y=0^-}^{\infty}\int_{z=0}^{\infty} Pr\{a^{n+1} \geq y+z-x / w^n = y,\ b_y^n = z\}\, d_z B(z/y)\, dW^n(y) =$$

$$= \int_{0^-}^{\infty}\int_{0}^{\infty} [1 - A(y+z-x)]\, d_z B(z/y)\, dW^n(y). \tag{4.3}$$

If the stochastic process $\{w^n\}$ is ergodic, then the limiting distribution function

$$W(x) = \lim_{n\to\infty} W^n(x), \tag{4.4}$$

exists and satisfies an integral equation

$$W(x) = \int_{0^-}^{\infty}\int_{0}^{\infty} [1 - A(y+z-x)]\, d_z B(z/y)\, dW(y). \tag{4.5}$$

We shall now discuss some properties of the solution of this equation.

*) Some results of [POSN 71] and [LIBU 71] are used in this section.

From (4.2) yields, that

$$W(x) = 0 \qquad \text{for} \quad x < 0 , \tag{4.6}$$

and from ergodicity of the process

$$W(0) = W_0 > 0 . \tag{4.7}$$

Let us assume additionally, that

a) $A(x)$ is continous for $x \in (-\infty, \infty)$,

b) $B(x/w)$ does not include an singular component.*)

We shall prove that under these assumptions the distribution function $W(x)$ is of the shape

$$W(x) = W_0 + \int_0^x v(y)\, dy , \qquad x \geq 0 \tag{4.8}$$

where $v(y)$ is a nonnegative function satisfying equation

$$\int_0^\infty v(y)\, dy = 1 - W_0 . \tag{4.9}$$

For the proof we shall asses, basing on (4.5), the differential quotient

$$\frac{W(x+h)-W(x)}{h} = \int_{0^-}^{\infty}\int_0^{\infty} \frac{A(z+y-x)-A(z+y-x-h)}{h}\, d_z B(z/y)\, dW(y) \tag{4.10}$$

for $x+h$, $x \in [0, \infty)$.

If $A(x)$ is continous for $x \in (-\infty, \infty)$ then

$$\exists_M \ \forall_{x, x-h} \quad \frac{A(x) - A(x-h)}{h} < M , \tag{4.11}$$

thus

$$\frac{W(x+h)-W(x)}{h} < \int_{0^-}^{\infty}\int_0^{\infty} M\, d_z B(z/y)\, dW(y) = M . \tag{4.12}$$

---

*) An singular function means a continous function with limited fluctuation, other then constant, having a derivative equal to zero almost everywhere.

So the discussed differential quotient is bounded, thus W(x) is continous in the interval $(0,\infty)$.

However (cf. [MARC 75]) W(x) , like any other distribution function may be rewritten us

$$W(x) = W_1(x) + W_2(x) + W_3(x), \tag{4.13}$$

where

$W_1(x)$ is a purely jump function,

$W_2(x)$ is a purely continous function,

$W_3(x)$ is a singular function.

It is clear that if B(x/w) does not contain a singular component then $W_3(x) = 0$.

On the other hand, continuity of W(x) in intervals $(-\infty,0)$, $(0,\infty)$ together with (4.6) and (4.7) lead to a conclusion that $W_1(x)$ has exactly one discontinuity point for $x = 0$. Utilizing the well known definition of pure continuity we obtain (4.8).

Functional equation (4.5) is difficult to solve directly in a general case. To show it more clearily let us rewrite this equation as

$$W(x) = \int_{0^-}^{x}\int_{0}^{x-y} d_z B(z/y)\,dW(y) + \int_{0^-}^{x}\int_{x-y}^{\infty} [1-A(y+z-x)]\,d_z B(z/y)\,dW(y) +$$

$$+ \int_{x}^{\infty}\int_{0}^{\infty} [1-A(y+z-x)]\,d_z B(z/y)\,dW(y). \tag{4.14}$$

Integrating by parts the second component of the right side of (4.14) and utilizing equalities

$$\int_{0}^{x-y} d_z B(z|y) = B(x-y/y) \quad \text{and} \quad B(0/y) = 0$$

we obtain

$$W(x) = \int_{0^-}^{x}\int_{x-y}^{\infty} A'(y+z-x)\,B(z/y)\,dz\,dW(y) + \int_{x}^{\infty}\int_{0}^{\infty} [1-A(y+z-x)\,d_z\,B(z/y)\,dW(y) \tag{4.15}$$

Differentiating both sides and utilizing equality

$$\frac{d}{dx}\left(\int_0^x g(x,y)\,dy\right) = g(x,x) + \int_0^x \frac{\partial g(x,y)}{\partial x}\,dy ,$$

we obtain

$$V(x) = -\int_{0^-}^{x} A'(0)B(x-y/y)\,dW(y) - \int_{0^-}^{x}\int_{x-y}^{\infty} A''(y+z-x)B(z/y)\,dz\,dW(y) +$$

$$+\int_{x}^{\infty}\int_{0}^{\infty} A'(y+z-x)\,d_z B(z/y)\,dW(y). \qquad (4.16)$$

The difficulties in analytical solution of this equation are caused by the last two components of the right side. It is clear that both this components vanish, if

$$A''(x) = -\lambda A'(x),$$

$$A'(x) = \lambda\,[1 - A(x)]. \qquad (4.17)$$

Conditions (4.17) hold only for an exponential distribution of interarrival time, $A(x) = 1 - e^{-\lambda x}$; and so far only this case was investigated*), leading to an functional equation

$$V(x) = \lambda\int_{0^-}^{x}[1 - B(x-y/y)]\,dW(y), \qquad (4.18)$$

with

$$W(x) = W_0 + \int_0^x V(y)\,dy ,$$

and W_0 defined from the normalization condition $\lim_{x\to\infty} W(x) = 1$.

It is worth mentioning that even (4.18) was not solved in the general case, but only for two special situations, listed below.

---

*) Authors of this paper succeded recently in solving equation 4.5 also for some other interarrival time distributions. The proper results will be published seperately.

Let us assume that the range of variation of w is devided into M + 1 intervals by introducing M+1 points w_i , such that

$$0 = w_0 < w_1 < \dots < w_i < w_{i+1} < \dots < w_M < w_{M+1} = \infty .$$

In [LIBU 71] it was assumed, that M=1 and

$$B(x/w) = \begin{cases} B_0(x) , & w = w_0 \\ B_i(x) , & w_{i-1} < w \leq w_i , \; i = 1,2 \end{cases} \qquad (4.19)$$

such that

$$E(b_i) = \int_0^\infty x \, dB_i(x) < \infty , \; i = 0,1 ,$$

$$E(b_1) = \int_0^\infty x \, dB_1(x) < \frac{1}{\lambda} .$$

The Laplace - Stieltjes transform of waiting time distribution function $\bar{w}(s)$ was found

$$\bar{W}(s) = \frac{\lambda W_0 [1 - \bar{b}_0(s)] - \lambda [\bar{b}_1(s) - \bar{b}_2(s)] \bar{\alpha}(s)}{s - \lambda [1 - \bar{b}_2(s)]} + W_0 , \qquad (4.20)$$

where

$$W_0 = \frac{1 - \lambda E(b_2)}{1 - \lambda [E(b_1) - E(b_0)] - \lambda [E(b_2) - E(b_1)] d} , \qquad (4.21)$$

$$d = \frac{1}{W_0} \int_0^{w_1} f(x) \, dx + 1 , \qquad (4.22)$$

$$\bar{\alpha}(s) = \int_0^{w_1} e^{-sx} f(x) \, dx , \qquad (4.23)$$

and $f(x)$ is a function identical with $w(x)$ for $0 < x < w_1$ having a Laplace- transform

$$\bar{f}(s) = \frac{\lambda W_0 [1 - \bar{b}_0(s)]}{1 - \lambda [1 - \bar{b}_1(s)]} . \qquad (4.24)$$

The probability generating function for queue length

$$\Pi(z) \stackrel{df}{=} \sum_{n=0}^{\infty} p_n z^n ,$$

where p_n is a probability of n demands waiting in the queue, was also found for this case

$$\Pi(z) = \frac{[\bar{b}_2(\gamma_z) - z\bar{b}_0(\gamma_z)]W_0 - z(\gamma_z)[\bar{b}_1(\gamma_z) - \bar{b}_2(\gamma_z)]}{\bar{b}_2(z) - z} , \qquad (4.25)$$

with $\gamma_z = \lambda(1-z)$.

Another special case was investigated in [POSN 73] where it was assumed, that

$$B(x/w) = 1 - e^{-\mu_i x}, \quad w_i \leq w < w_{i+1}$$
$$i = 0,1,2 \ldots M . \qquad (4.26)$$

The p.d.f. of waiting time w(x) and sojourn time u(x) where found, and defined piecewise as

$$w(x) = v_j(x) + W_0\delta(x),$$
$$u(x) = u_j(x), \qquad \text{for} \quad w_j \leq x < w_{j+1} \qquad (4.27)$$
$$j = 0,1,\ldots,M$$

where

$$v_0(x) = \lambda W_0 e^{(\lambda-\mu_0)x} ,$$

$$v_j(x) = \frac{\lambda W_0}{\mu_j - \mu_0 - \lambda}\left\{(\mu_j-\mu_0)e^{-\mu_0 x} - \lambda e^{(\lambda-\mu_j)x} \cdot e^{(\mu_j-\mu_0-\lambda)w_j}\right\} +$$
$$+ \sum_{r=0}^{j-1} \frac{\lambda \cdot K_r}{\mu_j - \mu_r - \lambda}\left\{(\mu_j-\mu_r)e^{-\mu_r x} - \lambda e^{(\lambda-\mu_j)x} e^{(\mu_j-\mu_r-\lambda)w_j}\right\},$$

$$u_j(x) = \sum_{r=0}^{j-1} \mu_r K_r e^{-\mu_r x} + \mu_j e^{-\mu_j x} K_j(x),$$

$$j = 1,2 \ldots M ,$$

with

$$K_o = W_o[\exp(\lambda w_1) - 1]$$

$$K_j = \frac{\lambda W_o}{\mu_j - \mu_o - \lambda}\left[e^{(\mu_j-\mu_o)w_{j+1}} - e^{(\mu_j-\mu_o-\lambda)w_j} e^{\lambda w_{j+1}}\right] +$$

$$+ \sum_{r=0}^{j-1} \frac{\lambda K_r}{\mu_j - \mu_r - \lambda}\left[e^{(\mu_j-\mu_r)w_{j+1}} - e^{(\mu_j-\mu_r-\lambda)w_j} e^{\lambda w_{j+1}}\right],$$

$$K_j(x) = \int_{w_j}^{x} e^{\mu_j y} v_j(y)\, dy, \qquad j = 1,2 \ldots M$$

while W_0 is found from the normalisation condition

$$W_0 + \sum_{i=0}^{M} \int_{w_i}^{w_{i+1}} v_i(x)\, dx = 1 .$$

5. Steady-state analysis, some special cases

In the previous section a methodology for analysis of a queueing system with waiting time dependent service time was given in the case of " open " queues of pure "delay" type. Thus we assumed, that the source of demands operates fully independently of the service process, and no limitations are imposed either on the queue length or possible waiting time.

In this section we shall discuss two special models, exceeding the framework of the above specified assumptions.

An attempt to describe a simple loss system was made in [MUDR 61] where a system $M/B_w/1$ with

$$B(x/w) = \Pi(x - \Psi(w)), \quad \Psi(w) = B_{max} - c \cdot w, \quad c > 0 \qquad (5.1)$$

was investigated.

Thus the service time was in this case a deterministic, decreasing function of waiting time. Additionally it was assumed, that waiting time is bounded by the value $w_{max} = \frac{B_{max}}{1+c}$ and every demand exceeding this waiting time leaves the system never to return.

Following formulas describing the mean service time and the loss probability P_L where derived

$$E(b) = B_{min} + \frac{1}{\lambda} e^{-\lambda w_{max}} + \frac{c}{\lambda}(1 - e^{-\lambda w_{max}}) , \qquad (5.2)$$

$$P_L = \frac{1}{\lambda E(b)} , \qquad B_{min} = B_{max} - c w_{max} . \qquad (5.3)$$

Another special case concerning "closed" queueing systems having an N dimensional source was analysed in [SWEN 59] . A $M/B_w/n/N/N/FIFO$ system was investigated for the case when $B(x/w)$ had the form

$$\bar{b}(s/w) = \frac{\mu}{s+\mu} \exp\left[-\lambda_d w \left(1 - \frac{\mu_d}{s+\mu_d}\right)\right] , \qquad (5.4)$$

with

$$\bar{b}(s/w) = \int_0^\infty e^{-sx} d_x B(x/w).$$

This very type of $B(x/w)$ resulted from the following model: The service of a demand which didn't have to wait consists of one phase which duration is a random variable b_1 , exponentially distributed with parameter μ , thus $b_1(x) = \mu e^{-\mu x}$. On the other hand the service of an demand which had to wait for time w is enlarged by addition of "k" consecutive phases, each of equally distibuted, random duration b_d.

It was assumed that k is a stochastic variable having Poisson distribution with parameter $(\lambda_d \cdot w)$ while b_d has an exponential distribution with parameter μ_d . Considering the

joint service time distribution one obtains directly formula (5.4)

As far as the generation process is concerned it was assumed that the time between completion of service and consecutive generation is for every demand random, exponentially distributed with parameter λ .

The above defined system was analysed by approximate methods. It was assumed that the mean service time of a single demand denoted $(\mu_i)^{-1}$ is dependent on the number of demands "i" actually generated /it means waiting in queue or being served / , and given by

$$\left.\begin{aligned} \mu_i &= \mu , && i \leq n \\ n\mu_i &= n\mu - (i-n)\Theta\lambda_d , && N \geq i > n \end{aligned}\right\} , \qquad (5.5)$$

where $\Theta = \frac{\mu_d}{\mu}$.

As the approximation of the original case a state dependent system having exponentially distributed service time with parameter μ_i /defined by (5.5)/was considered.

Such a system is ergodic if

$$n\mu > (N-n)\Theta\lambda_d . \qquad (5.6)$$

It can be easily shown that p_i - the probabilities of "i" demands being actually generated are, in the steady-state given by

$$\begin{aligned} p_0 &= \Big[\sum_{i=1}^{n-1} \Big(\frac{\lambda}{\mu}\Big)^i \binom{N}{i} + \sum_{i=n}^{N} \gamma_i\Big]^{-1} \\ p_i &= p_0 \Big(\frac{\lambda}{\mu}\Big)^i \binom{N}{i} , && i < n \\ p_i &= p_0 \gamma_i && N > i \geq n \end{aligned} \qquad (5.7)$$

where

$$\gamma_i = \Big(\frac{\lambda}{\mu}\Big)^i \frac{(N)_i}{n!\, n^{i-n}} \Big\{\prod_{k=0}^{i-1}\Big(1 - \frac{k\Theta\lambda_d}{n\mu}\Big)\Big\}^{-1} ,$$

$$(N)_i = \frac{N!}{(N-i)!} .$$

Basing on the values of p_i one can directly obtain all system's parameters , like mean waiting time, mean queue length, etc. Proper formulas have been derived and the obtained results were validated by simulation. However because of very short simulation runs no meaning estimations of the accuracy of this approximation were obtained.

6. The effect of queue discipline on system's throughput

So far we constrained our considerations to the FIFO queueing discipline. However, as in the investigated queueing systems the service time of every individual demand depends on it's waiting time, it can be expected that the queue discipline will influance not only the queue length or waiting time distribution, like in the classical cases, but also the system throughput.

For the sake of further considerations we shall constrain ourselves to a special case of system $M/B_w/1$ with

$$B(x/w) = \begin{cases} B_1(x) = \Pi(x - B_s) , & w \leq D \\ B_2(x) = \Pi(x - B_L) , & w > D , \end{cases} \qquad (6.1)$$

which will be referred to as a "basic system".

In [BUZA 74] a basic system with $D = B_s$ was investigated using the imbedded Markov chains approach, for the description of system state in time epochs, immediately following completion of consecutive services.

Following scheduling disciplines were considered:

I. Disciplines without input forecast

a/ FIFO - first come first served

b/ LIFO - last come first served

II. Disciplines with input forwcast

It has been assumed that,although the input stream is Poisson an information is available wheather in following x time units the next demand will arrive / a positive forecast / or not /a negative forecast /. If after completion of consecutive service the considered system is not empty,and the next demand chosen according to the scheduling rules needs a "short" service /that means service of duration B_s/ then the forecast is neglected,and the next service starts. However if the next demand needs a "long" service /of duration B_L / then the forecast will be considered.In the case of positive forecast the server remains idle until the new demand arrives,and starts immediately it's service upon arrival. In the case of negative forecast the demand chosen for service is given due attention.

As basic scheduling rules for choosing the next demand for service, FIFO and LIFO were considered,which after introducing the forecast will be denoted as

a/ FIFO(x)

b/ LIFO(x)

It is obvious that for $x = 0$ disciplines IIa/ and IIb/ become Ia/ and Ib/ respectively.

For the above defined scheduling disciplines, proper formulas defining its maximal throughput have been obtained,and listed in Table 1.

Let us notice that the formulas should be understood as follows: If T_{DISC} satisfies the below given equations for scheduling discipline DISC,then for every $\lambda < T_{DISC}$ the considered system

with this scheduling discipline will be ergodic.

DISC	
FIFO	$T B_L = 1$,
LIFO	$T B_S + T(B_L - B_S) e^{-T B_S} = 1$,
FIFO (x)	$T B_S + 1 + e^{-Tx} T(B_L - B_S - x) - e^{-Tx} = 1$, $0 \leq x \leq B_L$
LIFO(x)	$T B_S + e^{-Tx} e^{-T B_S} T(B_L - B_S - x) + e^{-T B_S}(1 - e^{-Tx}) = 1$, $0 \leq x \leq B_L - B_S$ $T B_S + e^{-T B_S}(1 - e^{-Tx}) + e^{-T B_L}(1 - T B_S) - e^{-T B_S} e^{-Tx}[1 - T(B_L - x)] = 1$, $B_L - B_S \leq x \leq B_L$

Table 1.

It is easy to prove, basing on the above listed formulas, that:

$$\max_{x \in [0, B_L]} T_{DISC(x)} = T_{DISC(B_L - B_S)}, \qquad (6.2)$$

$$\forall_{x \in (0, B_L]} \; T_{DISC(x)} > T_{DISC(0)}, \qquad (6.3)$$

where DISC(x) = FIFO(x) , LIFO(x) ;

$$\forall_{x, y \in [0, B_L]} \; T_{LIFO(x)} > T_{FIFO(y)} . \qquad (6.4)$$

Equation (6.2) defines the optimal forecast length, while inequality (6.3) shows that any forecast always increases system's throughput. From inequality (6.4) it becomes clear that any LIFO scheduling discipline is always better /from the point of view of the throughput / than any FIFO discipline. Quantitative data for throughput comparison under various scheduling rules /taken from [BUZA 74] / are given in Table 2 for the case when $B_L = 2B_S$.

DISC	$T'B_s$
FIFO	0.500
FIFO(B_L-B_S)	0.568
LIFO	0.659
LIFO(B_L-B_S)	0.751

Table 2

Basing on this results one can state, that the scheduling discipline significantly influances maximal throughput in queueing systems with waiting time dependent service times.
In the considered case, contradictory to classical queueing systems, keeping the server idle in presence of a nonempty queue may increase system's throughput.

The results presented in [BUZA 74] make it possible to compare some above described disciplines. However these results do not lead to pointing out an optimal /from the point of view of the throughput / discipline in some larger class of scheduling disciplines. The problem of proving optimality was considered in [WOLI 79] where for the basic system /with $B_S \leq D \leq B_L$ / it was proved that among nonpreemptive disciplines:

a/ disciplines where starting of "long" service is permitted, although there are in the queue some demands needing "short" service, have always lower maximal throughput than disciplines where such a sequence is impossible,

b/ among disciplines where server's idleness in the case of nonempty queue is not permitted, optimal is the following discipline:

Demands needing "short" service obtain priority over those needing "long" one, and should be served according to the order of arrival,

c/inserting idleness of the server when all demands waiting in the queue need "long" service increases the system's throuput,

d/There exists an optimal forecast length,

e/the optimal nonpreemptive discipline is similar to that given in b/ with respect to demends needing "short" service.If there are no such demands, and the forecast for optimal time period is positive, the server should remain idle,and if negative than it should start a "long" service.

Preemptive disciplines , granting "short" services higher priority, allow theoretically for increasing of system's throughput. However in the majority of applications only nonpreemptive scheduling is permitted .

7. Examples

After considering the effect of scheduling discipline on system throughput we shall now return to the FIFO queues.

The objective of this section is to answer a following question:

Are systems with waiting time dependent service time essentially different, from the point of view of their main parameters then, say, classical M/G/1 systems having identical unconditional service time distribution.Putting it in different words one could ask, is it really necessary to overcome a number of difficulties in order to obtain an exact solution of such system, or perhaps parameters of an appropriatly chosen M/G/1 system would serve as a good enough approximation of the original case.

For this comparison we shall use again the basic system as defined by (6.1), being in fact a special case of the system given by (4.19). For the purpose of demonstrating the methodology of system analysis, being a continuation of general considerations presented in section 4, we shall solve this example in detailes, applying a method suggested in [LIBU 71].

Utilizing (6.1) in (4.18) we obtain for $x < D$

$$v(x) = \lambda W_0 [1 - B_1(x)] + \lambda \int_0^x v(y)\,dy - \lambda \int_0^x B_1(x-y)\, v(y)\,dy . \qquad (7.1)$$

Let us introduce a function $f(x)$ satisfying (7.1) for every x, and identical with $v(x)$ for $0 \leqslant x < D$.

After transformation we obtain

$$\bar{f}(s) = \frac{1}{s}\lambda W_0 [1 - \bar{b}_1(s)] - \frac{\lambda}{s} f(s) + \frac{\lambda}{s} \bar{b}_1(s) f(s) , \qquad (7.2)$$

thus

$$\bar{f}(s) = \frac{\lambda W_0 [1 - \bar{b}_1(s)]}{s - \lambda [1 - \bar{b}_1(s)]} , \quad \bar{f}(s) = \int_0^\infty e^{-st} f(t)\,dt . \qquad (7.3)$$

Let us introduce also

$$\Phi(x) = W_0 + \int_0^x f(t)\,dt , \qquad (7.4)$$

identical with the distribution function $W(x)$ for $0 \leqslant x < D$.

Denoting

$$\bar{\varphi}(s) = \int_{0-}^\infty e^{-st}\, d\Phi(t)$$

and utilizing (4.15) we obtain

$$\bar{\varphi}(s) = W_0 + \bar{f}(s) = \frac{W_0 \cdot s}{s - \lambda [1 - \bar{b}_1(s)]} . \qquad (7.5)$$

However (6.1) yields

$$\bar{b}_1(s) = e^{-sB_s},$$

$$\bar{\varphi}(s) = s W_0 \sum_{j=0}^{\infty} \frac{e^{-sjB_s}}{(\lambda - s)^{j+1}}, \tag{7.6}$$

thus after invers transformation, with respect to continuity of $\Phi(x)$ for $x > 0$ we obtain

$$\Phi(D) = W_0 \cdot d \tag{7.7}$$

where

$$d = \sum_{j=0}^{n} e^{\lambda(D - jB_s)} \frac{[-\lambda(D - jB_s)]^j}{j!} \tag{7.8}$$

$$\text{for} \quad nB_s \leqslant D < (n+1)B_s.$$

On the other hand (4.18) can be rewritten as

$$V(x) = \lambda W_0 [1 - B(x/0)] + \lambda \int_0^x V(y)\,dy - \lambda \int_0^x B(x-y/y) V(y)\,dy. \tag{7.9}$$

After transformation, some rearrengements and utilization of (6.1) we obtain

$$\bar{V}(s) = \frac{\lambda W_0 [1 - \bar{b}_1(s)] - \lambda [\bar{b}_1(s) - \bar{b}_2(s)] \bar{\alpha}(s)}{s - \lambda(1 - \bar{b}_2(s))}, \tag{7.10}$$

where

$$\bar{V}(s) = \int_{0_D}^{\infty} e^{-st} V(t)\,dt,$$

$$\bar{\alpha}(s) = \int_0^D e^{-st} f(t)\,dt.$$

We are now in the position to derive W_0 as

$$W_0 + \lim_{s \to 0} \bar{V}(s) = 1, \tag{7.11}$$

where

$$\bar{b}_1'(0) = -B_s, \quad \bar{b}_2'(0) = -B_L,$$

$$\bar{\alpha}(0) = \int_0^D f(x)\,dx = \Phi(D) - W_0 = W_0(d-1),$$

and finally

$$W_0 = \frac{1 - \lambda B_L}{1 - \lambda(B_L - B_s)d}. \tag{7.12}$$

The probability of consecutive service to be a "short" one /of duration B_S / is equal to

$$W_\lambda(D) = \Phi(D) = \frac{1-\lambda B_L}{d^{-1}-\lambda(B_L-B_S)} \,. \tag{7.13}$$

Let us now discuss $B_\lambda(x)$, the unconditional service time distribution being evidently dependent on the intensity of arrivals λ,

$$B_\lambda(x) = \int_{0^-}^{\infty} B(x/y)\, dW(y). \tag{7.14}$$

Utilizing (7.13) we obtain

$$B_\lambda(x) = W_\lambda(D)\, 1(x-B_S) + [1-W_\lambda(D)]\, 1(x-B_L), \tag{7.15}$$

thus

$$E(b_\lambda) = W_\lambda(D)\, B_S + [1-W_\lambda(D)]\, B_L \,,$$

$$E(b_\lambda^2) = W_\lambda(D)\, B_S^2 + [1-W_\lambda(D)]\, B_L^2 \,.$$

We shall now derive the mean waiting time $V(\lambda)$

$$V(\lambda) = E(w) = E(v) = \lim_{s\to 0} (-\bar{v}'(s)) \,. \tag{7.16}$$

As both the numerator and denominator of $\bar{v}(s)$ tend to zero with $s\to 0$, utilization of the de l'Hospital rule is necessary. After laborious derivations and utilizing the equality

$$\int_0^T e^{\lambda x} \frac{(-\lambda x)^j}{j!}\, dx = \frac{e^{\lambda T}}{\lambda} \sum_{i=0}^{j} \frac{(-\lambda T)^i}{i!} - \frac{1}{\lambda} \tag{7.17}$$

we obtain

$$V(\lambda) = \frac{\lambda B_L^2}{2(1-\lambda B_L)} - \lambda(B_L-B_S)\, \frac{2P+(B_L+B_S)d}{2[1-\lambda(B_L-B_S)d]} \,, \tag{7.18}$$

with

$$P = dD + \frac{1}{\lambda}\left\{n+1-\sum_{j=0}^{n}\sum_{i=0}^{j} e^{\lambda(D-jB_S)}\, \frac{[-\lambda(D-jB_S)]^i}{i!}\right\},$$

where n is defined from the inequality $nB_S \le D < (n+1)B_S$.

The above presented analysis is valid for the steady-state. Basing on theorems presented in section 3 it is easy to varify that the considered system is ergodic for arrival intensity $\lambda < T$:

$$T = \frac{1}{B_L} . \tag{7.19}$$

After completing analysis of the basic system we shall now compare it's parameters with an "equivalent" M/G/1 system which service time distribution is given by (7.15) .Thus in both systems the service times have identical distribution function,but in the "equivalent" system service times are assumed to be independent which is not true in the basic system.

Such an "equivalent" system could be used for estimating the parameters, /and perhaps forecasting the behaviour under different load/ of an real - life basic system for which the distribution function $B_{\lambda_o}(x)$ would be experimentally found through direct measurements,for some intensity of arrivals λ_o .

We shall start this comparison with calculation of the maximal throughput in both systems.

For the basic system we have

$$T_B = \frac{1}{B_L} ,$$

while for the "equivalent" system the maximal throughput $T_p(\lambda_o)$ can be defined as

$$T_p(\lambda_o) = [E(b_{\lambda_o})]^{-1}, \tag{7.20}$$

where λ_o is the arrival intensity used for defining $B_{\lambda_o}(x)$. The obtained from proper calculations $T_p(\lambda_o)$ versus λ_o is plotted in figures 1,2,3 for various values of parameters B_L, B_S, D of the basic system.

It is visible that always

$$T_p(\lambda_o) > T_B \tag{7.21}$$

and the difference $R(\lambda_o)=T_p(\lambda_o)-T_B$ decreases when λ_o increases. Additionally

$$\lim_{\lambda_o \to 0} R(\lambda_o) = \frac{1}{B_S} - \frac{1}{B_L} , \tag{7.22}$$

$$\lim_{\lambda_o \to T_B} R(\lambda_o) = 0 . \tag{7.23}$$

The function $R(\lambda_o)$ is a strictly decreasing and nonlinear one. Initially, for small λ_o it has almost constant values, and decreases rapidly for λ_o tending to T_B.

Thus using the "equivalent " system as an approximation of the basic system would cause significant error in the evaluation of system's maximal throughput.

Some numerical data for $B_L = 0.1$, $B_S = 0.05$, $D = 0.1$ describing the relative error in evaluation of the maximal throughput $\frac{R(\lambda_o)}{T_B}$, the utilization factor $\frac{\lambda_o}{T_B}$ and the estimated utilization factor $\frac{\lambda_o}{T_p(\lambda_o)}$ are given in Table 3.

λ_o	2.	4.	6.	8.
λ_o/T_B	20%	40%	60%	80%
$R(\lambda_o)/T_B$	99.99%	99.48%	97.26%	87.76%
$\lambda_o/T_p(\lambda_o)$	10%	20%	30.4%	42.6%

Table 3

For example if the utilization factor for the basic system $\frac{\lambda_o}{T_B} = 0.8$ then the estimation based on the "equivalent" system would lead us to conclusion that the utilization factor is

as low as 42.6% . In fact 25 % increase of λ_o would exceed systems' maximal throughput.

Let us now compare mean waiting times in both discussed systems. For the basic system, the mean waiting time $V(\lambda)$ was defined by (7.16), while for the "equivalent" system the mean waiting time denoted $U(\lambda,\lambda_o)$ may be found from the well-known formula describing the M/G/1 system with respect to (7.15) as

$$U(\lambda,\lambda_o) = \frac{\lambda E(b^2_{\lambda_o})}{2[1-\lambda E(b_{\lambda_o})]} =$$

$$= \frac{\lambda[W_{\lambda_o}(D)(B_S^2 - B_L^2) + B_L^2]}{2[1-\lambda[W_{\lambda_o}(D)(B_S - B_L) - B_L]]} . \qquad (7.24)$$

Values of $V(\lambda)$ and $U(\lambda,\lambda_o)$ versus λ are plotted in figures 4 and 5 for two sets of parameters B_L, B_S, D .

Comparing values $V(\lambda)$ and $U(\lambda,\lambda)$ one can asses the differance of mean waiting time in the basic system and mean waiting time estimated from the "equivalent" system for some arrival intensity λ. Curves $U(\lambda,\lambda_o)$, $\lambda_o \neq \lambda$ give the estimation of mean waiting times for different λ, obtained basing upon an "aquivalent" system with service time distribution function $B_{\lambda_o}(x)$.

It is easy to see that the estimation error $V(\lambda) - U(\lambda,\lambda_o)$ is negative for small λ and monotonically increases with the increase of going through zero for some $\lambda_{\lambda_o} < \lambda_o$.

From the above given computational examples it becomes obvious that using simple M/G/1 models for estimation of parameters of queueing systems with waiting time dependent service times is not desirable as cousing significant errors.

8. Final remarques .

Queueing systems with waiting time dependent service times have to be studied, because of their significance in description of numerous real-life processes.

As it has been shown in this paper, neglecting this dependence and estimating parameters of this system by the use of appropriatly chosen classical models, leads to essential errors not only quantitative but also qualitative.

On the other hand, analysis of such systems is difficult and so far only a limited number of results, discussed in this paper have been achieved. A lot of interesting questions remain still unanswered among which the following ones seem to be specially important:

- effect of the type of distributions A(x) and B(x/w) on system's parameters,
- optimization of queueing disciplines,
- analysis of batch service systems,
- multiserver queueing systems features.

Because of the need of such results for application purposes it can be expected that more effort will be done in this area and that the knowledge about the discussed here new class of queueing systems will be in near future significantly enlarged.

A.Wolisz, Ph.D
J.Izydorczyk, M.Sc

Polish Academy of Sciences
Department of Complex Automation
44-100 Gliwice, Zwycięstwa 21,
POLAND

References

[BUZA 74] J.A.Buzacott "The Effect of Queue Discipline on the Capacity of Queues with Service Time Dependent on Waiting Times" INFOR 12, 1974 ,pp 174-185

[CALL 73] J.R.Callahan "A Queue with Waiting Time Dependent Service Times" NRLQ 20, 1973 ,pp 321-324

[GROS 74] D.Gross,C.M.Harris "Fundamentals of Queueing Theory" John Wiley,New York,1974

[KLEI 75] L.Kleinrock "Queueing Systems, Vol.I :Theory ", John Wiley,New York,1975

[LIBU 71] M.Libura "On a Single Server Queueing System with Service Time Dependent on Waiting Time" Archiwum Automatyki i Telemechaniki 16, 1971 ,pp 279-286 /in Polish /

[MARC 75] B.G.Martsenko et al. "Linear Stochastic Processes and Their Applications " Naukova Dumka ,Kiev, /in Russian/

[MUDR 61] V.I. Mudrov "Queueing with Impatient Customers and Variable Service Time,Linearly Dependent on the Queueing Time of the Customer" Problemy Kibernetiki 5, 1961 , pp 283-385 /in Russian/

[POSN 73] M.Posner "Single Server Queues with Service Time Dependent on Waiting Time" Op.Res 21, 1973 ,pp 610-616

[SUGA 65] S.Sugawara,M.Takahashi "On Some Queues Occuring in an Integrated Iron and Steel Works " J of O.R. Soc of Japan 8, 1965 ,pp 16-23

[SWEN 59] O.Swenson "An Approach to a Class of Queueing Problems" Op.Res 7, 1959 ,pp 276-292

[TWEE 75] R.L.Tweedie "Sufficient Conditions for Ergodicity and Recurrence of Markov Chains on a General State Space" Stochastic Processes and Their Applications 3, 1975 , pp 385-403

[WOLI 79] A.Wolisz "Optimal Scheduling Disciplines for a Class of Single Server Queues with Waiting Time Dependent Service Times " Research Report,Department of Complex Automation Systems,Polish Academy of Sciences,Gliwice

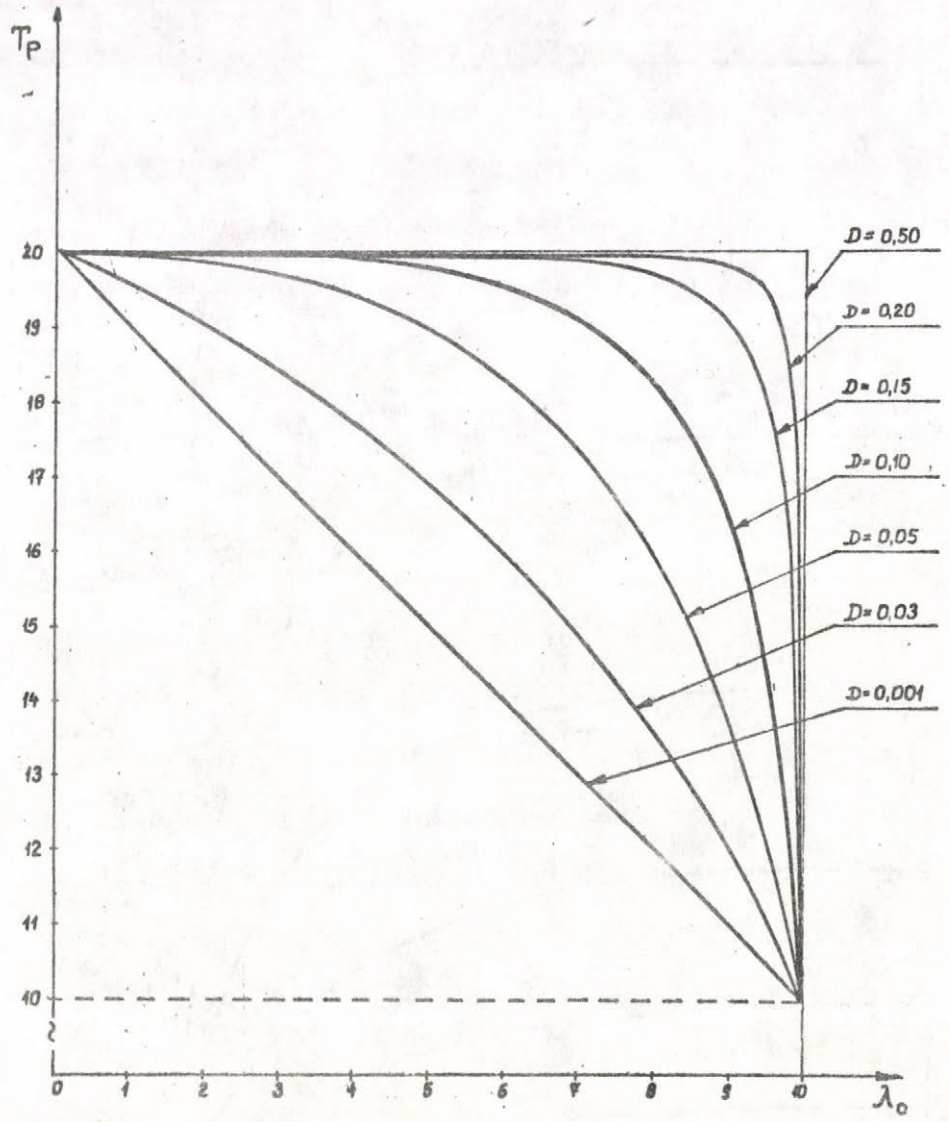


Figure 1 . Throughout of the "equivalent" system versus λ_0 for $B_L = 0.1$, $B_S = 0.05$

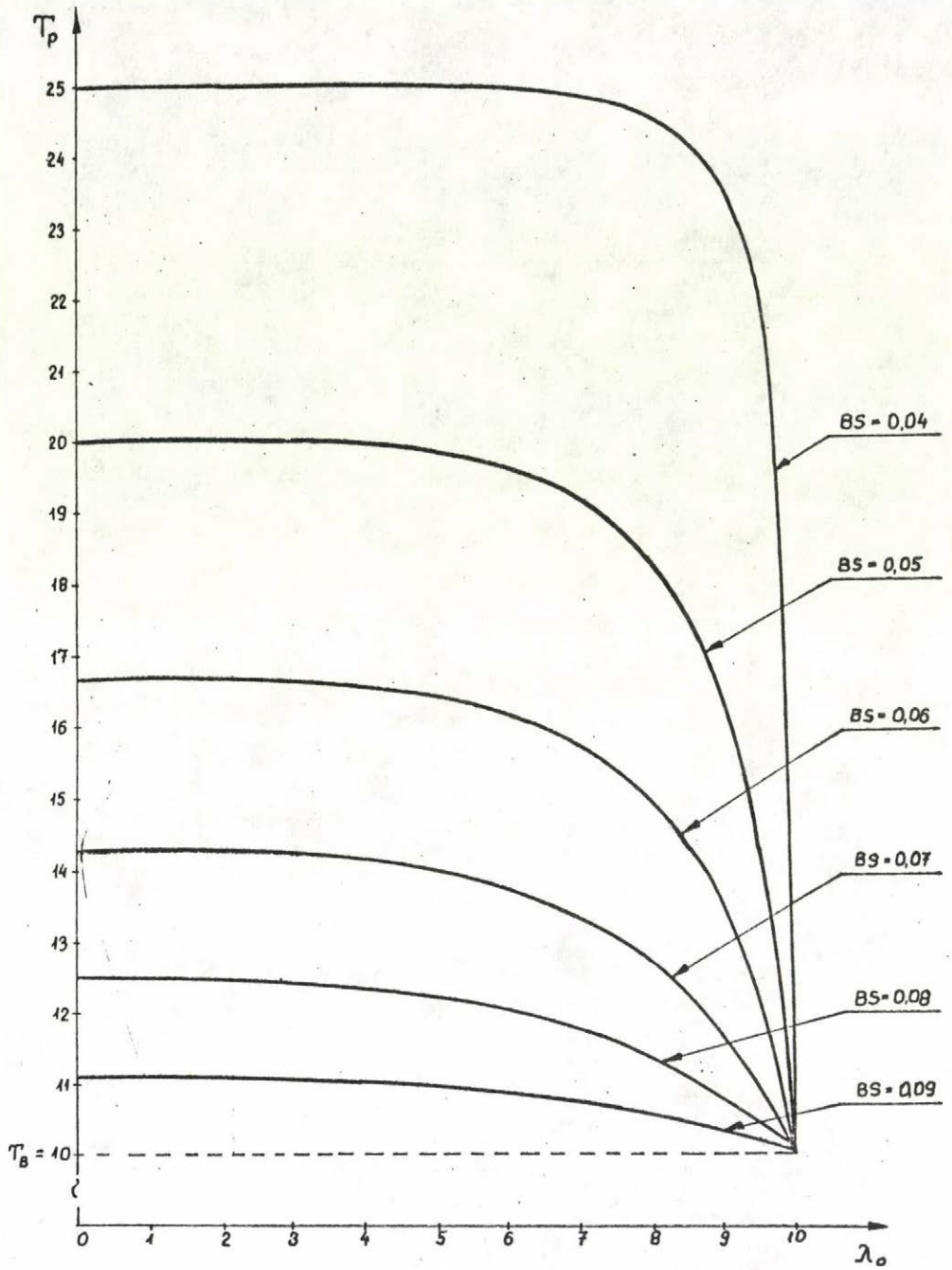


Figure 2 . Throughput of the "equivalent" system versus λ_0 for $B_L = 0.1$, $D = 0.1$

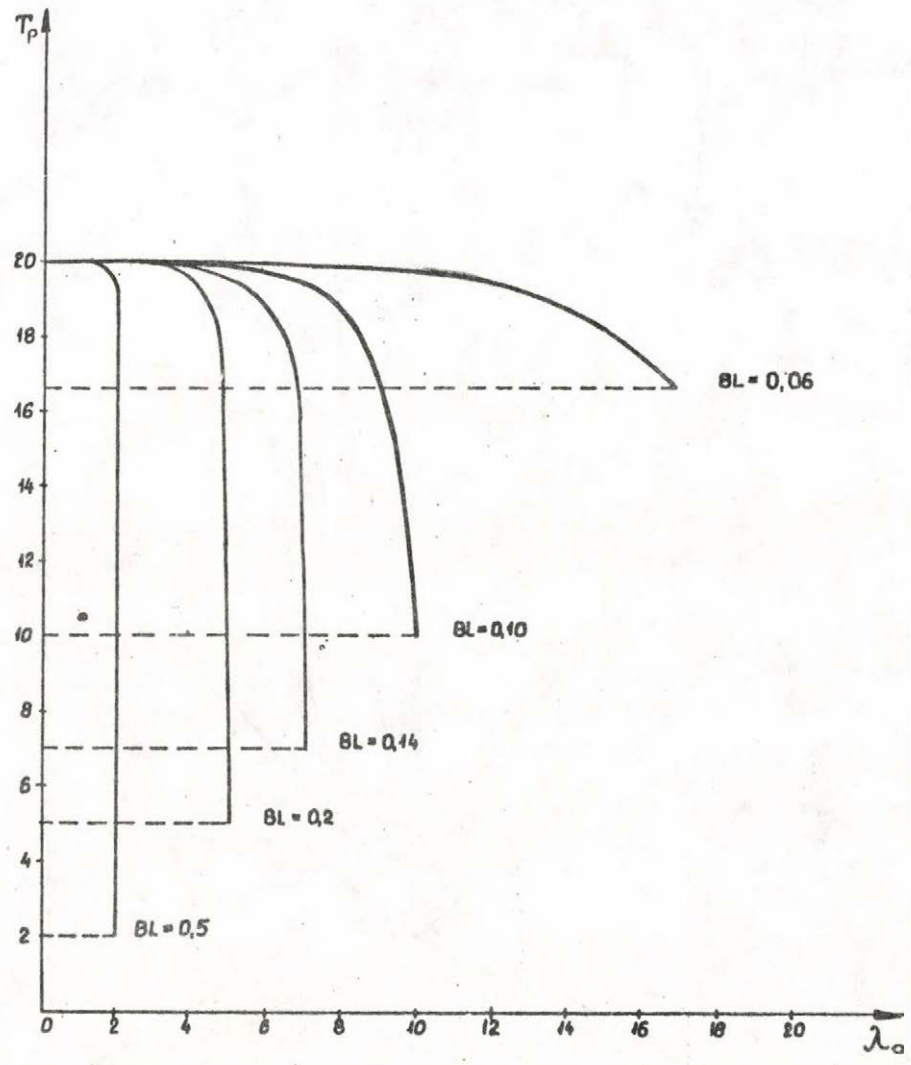


Figure 3 . Throughput of the "equivalent" system versus λ_a for $B_S = 0.05$, $D = 0.1$

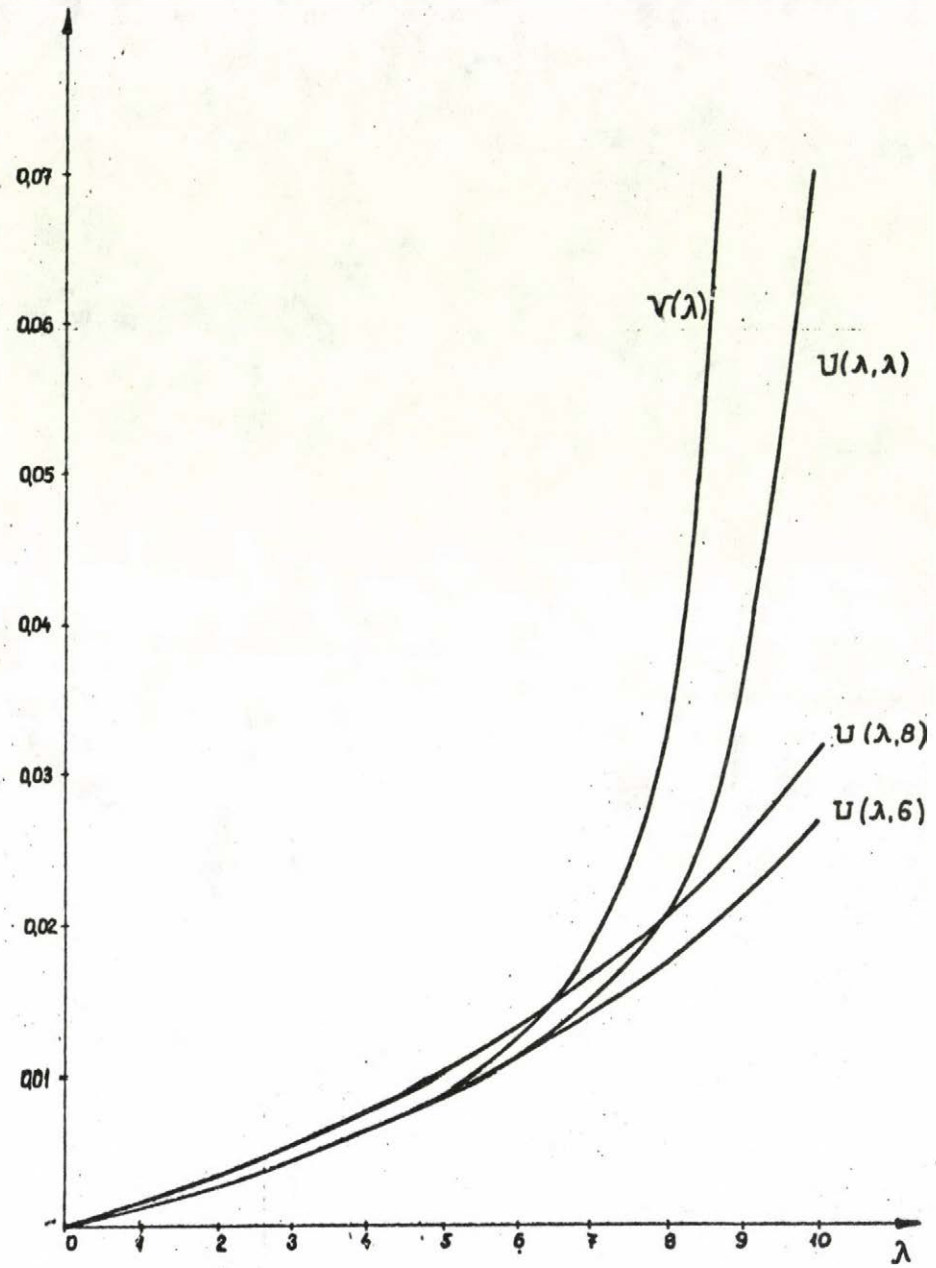


Figure 4 . Mean waiting time in the basic system and "equivalent" system versus λ for $B_L = 0.1$, $B_S = 0.05$, $D = 0.1$.

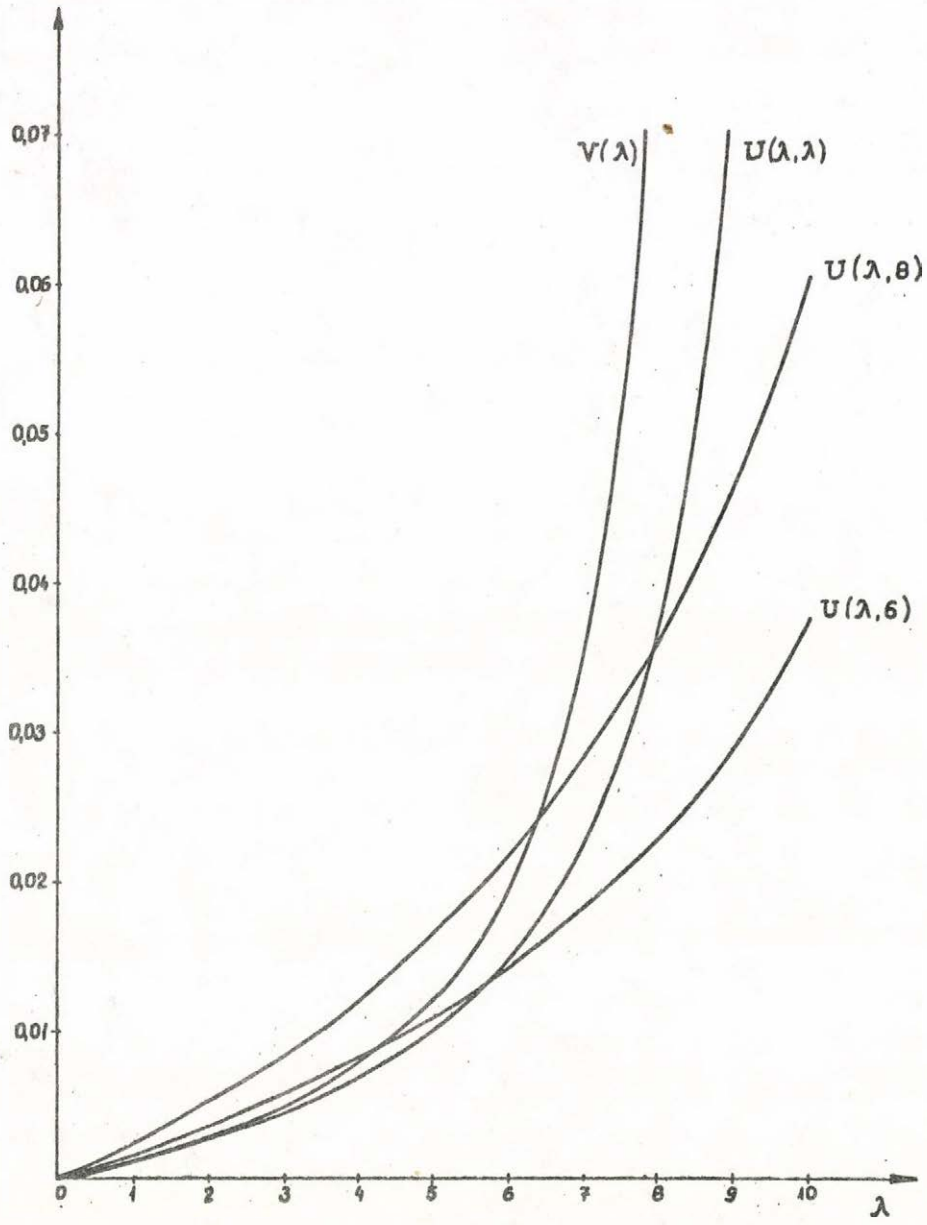


Figure 5 . Mean waiting time in the basic system and "equivalent" system versus λ for $B_L = 0.1$, $B_S = 0.05$, $D = 0.05$

SIMSCRIPT SIMULATION MODEL FOR COMPUTER SYSTEM EVALUATION

Dr. S. Zárda - A. Nagy
Computer Service of State Administration
Budapest, Hungary

ABSTRACT: The paper presents a discrete event-driven stochastic simulation model written in SIMSCRIPT simulation language for a Honeywell-Bull computer system under GCOS /General Comprehensive Operating Supervisor/ control. The model has been developed particularly for elaborating an effective running strategy by setting the suitable parameters of the system startup deck and finding a bottle-neck. The model is named HwB-SIM.

INTRODUCTION

Simulation techniques have more and more applications in the fields of multiprogrammed computer system performance evaluation. Simulation models enable analysts to predict how systems will behave under a variety of operating conditions. Model applications are advisable where performance is being tuned by varying parameters. Thus the model makes it possible for a nalysts to evaluate systematically a wide range of possible parameter settings and determine which settings are optimal for the system being tuned. This type of models is

called performance simulator. There is a large number of commercially available performance simulators e.g. BEST/1, CASE, IMSIM, SAM, SCERT 76. /1/ The simulators differ according to the employed mathematical techniques based on queuing theory or a traditional event-driven simulator driven by random number generators and detailed workload descriptions. General opinion is that the step by step simulation without applying the queuing theory is considerably less efficient than the other type. But there is a large number of particular tuning problems where the event-driven simulator can be extremely useful. HwB-SIM has been developed only for a given Honeywell Computer structure. This doesn't exclude however its application for another structure of Honeywell system, even for other types of computers. This derives from the fact that the modelling phase is the same, and the basis of modelling process is the analogy between elements of the real system and the blocks of the model. The main purpose of our model creation was to increase the total throughput of given computer system assuming a given job-mix. Through increasing of throughput we can achieve an increase in the income of our computer center. After analyzing simulation results we can try to tune our system so that idle time of any device is not allowed and there are no unused capacities.

Summarizing the problem definition: building up a simulation model, that simulates a multiprogrammed time-sliced operating

computer system in batch-processing environment, assuming total workload. The purpose of this research is to set the suitable parameters of startup deck and to choose the right priority strategy of operating system modules in order to maximize the capacity utilization.

Model type and level

We may compose HwB-SIM as a general purpose queuing model with multiple channels and multiple facilities. If an arrival may enter anyone of several facilities, the group of facilities is called a channel. And when the departures of one channel enter another channel, we have a case of multiple channels. When the output of one station becomes the input of another station, we have a situation known as multiple facilities. HwB-SIM is basically a stochastic model although we made a lot of efforts eliminating this type of effects. We defined the attributes of elements following the empiric distributions.

One of the most important factors of any performance simulation project is choosing the proper level of detail for representing the system and the workload. If the representation is not detailed enough, the model will not be able to produce accurate predicions consistently. On the other hand, if the representation is too detailed, the simulation model may be sensitive to parameters which vary from day to day and are difficult to estimate precisely. A good simulation model produces precise predictions on the basis of a small set of para-

meters that are reasonably stable and can be estimated with confidence. Another important point of modeling is the model's cost which also depends on the detail level. We have chosen functional modelling. It is not based on the microlevel characteristics of a specific device /e.g. execution time of one instruction/ but reflects the procedure characteristics of one logical unit of model /e.g. a job or job-step /activity/ /. A job is represented by 6 attributes and an activity /job-step/ is represented by 19 attributes.

Arrival process /inflow/

The incoming requests are defined as an inflow process. The inflow process is dynamic if the requests in the /0,t/ time-interval arrive continuosly according to some distribution function. In HwB-SIM the incoming requests mean reading of jobs, so our inflow process is dynamic. Another factor of inflow process is the sequence of the incoming jobs. This sequence determines the job-mix in a given time. More precisely, in a given time the job-mix is composed by the requests of the cyclic arrivals and the new arrivals /see later in Figure 6./ . The cyclic arrivals are handled by the system's own algorithm, and the new arrivals by their time-sequence. This ensures random arrivals.

Model input /sampling techniques/

The task of model is the treatment of jobs and activities.

When a request is flowing through the model we have to know the sequence of servicing, the unique service times, the requirements of servicing e.t.c.. These values are based on two groups of data:

- job control cards information
- SCF /Statistical Collection File/ created by GCOS for accounting purposes.

The data base of the simulation model contains values sampled on a randomly chosen day. It contains 81 jobs that are devided to 196 activities. From this set 20 jobs /57 activities/ were selected following the rule of layered sampling techniques which were run through the system with different settings of parameters and alternatives of priority strategies. The sample reflects very well the avarage daily workload and the avarage job-mix. The main characteristics of the chosen job-mix are: the job-stream consists of COBOL, IDS, GMAP, FORTRAN, UTILITY and other programs, as listed in Figure 1.

Job type	Number of jobs	Number of activities	% of total
COBOL-IDS	8	19	40
GMAP	4	14	20
FORTRAN	4	9	20
UTILITY, FILSYS, other	4	15	20

Figure 1. Representative job-stream

The jobs consist of 2,8 a ctivities in average. The actual distribution is shown in Figure 2.

Number of activities	Number of jobs
1	3
2	7
3	7
6	2
7	1

Figure 2. Activity distribution in ters of jobs

The avarage core requirement was 22,49 K words, the actual distribution is shown in Figure 3.

Required core-size /word/	Number of activities
10-12 K	9
16 K	11
20 K	5
22 K	3
24 K	7
25-28 K	7
32 K	13
40 K	2

Figure 3. Core-requirement distribution

The avarage execution time /CPU time/ was 62,8 seconds per activity.60 percent of sample derived from card-reader, 20 percent from magnetic tape and other 20 percent were "spown job" from disc. The output destination distribution was as follows: 80 percent of sample went to line-printer, 10 percent to tape and 10 percent to disc.

Model description: components and operating

HwB-SIM contains two large groups of resources: the hardware and the software capacities.

The simulated hardware configuration is the following:

- central processor system with 96K words of main memory
- peripherals:

  card reader /CRU 1050/

  line printer /PRU 1200/

  magnetic tape subsystem with four tape units /MTU 400/

  disc storage subsystem with two disc units /MSU 400/

The outline structure of HwB 66/20 computer system configuration is shown in Figure 4.

Service times of hardware units are based on the Honeywell technical descriptions, e.g. speed of card reader is 1050 cards/second, e.t.c. The modelling of channel network is based on empirical observations. At maximal channel workload one card reader, one line printer, one tapedriver and two disc units can operate simultaneously.

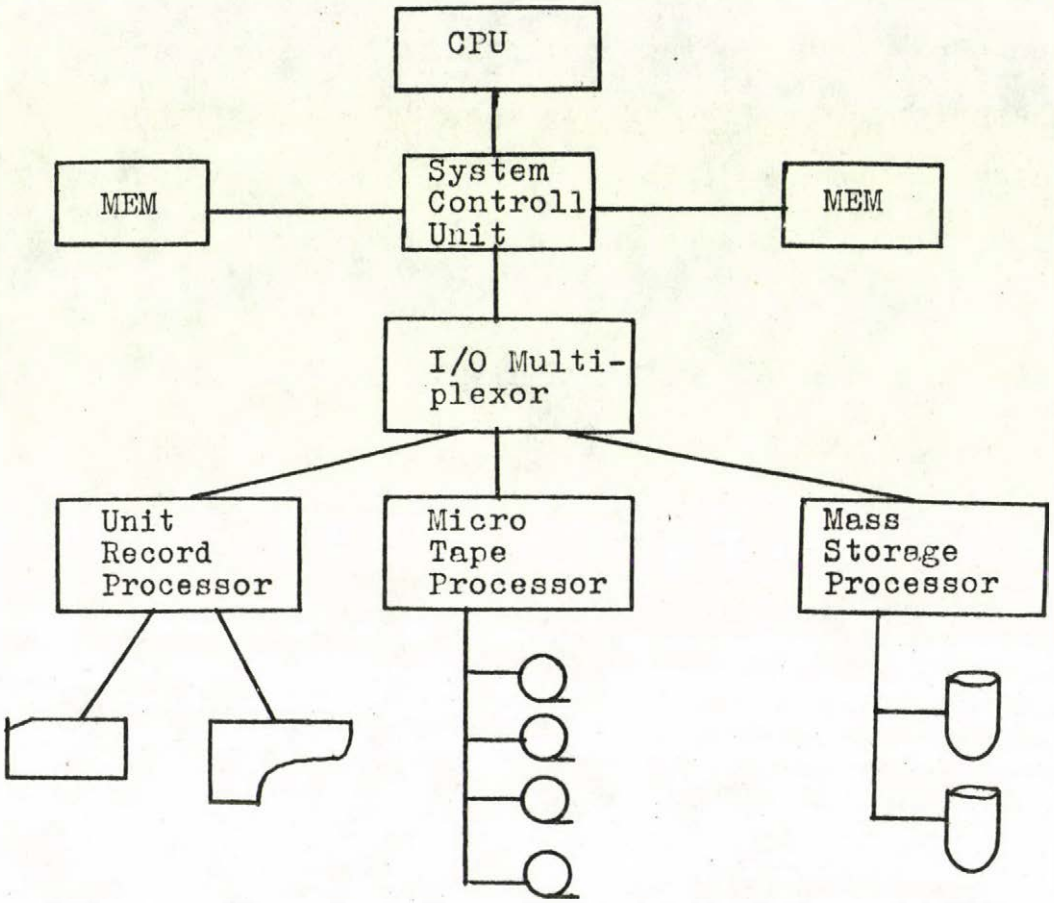


Figure 4. Simulated hardware configuration

The software capacities are the modules of GCOS. The control programs of GCOS compose the HwB-SIM guideline. The tasks of GCOS modules are separated well. In some modules we have possibilities to intervent the system. These points are named as decision poiñts.

SYSTEM INPUT: It begins to operate if an arrival occurs.

It operates as a normal program with a special priority. There are three different types of it according to the input media. At the same time more than one may exist /limited only by the memory space/. System input classifies the jobs according to their requirements. It builds up three groups of queues: normal, express, hold, and forwards the jobs to scheduler's input queue. At this decision point we can alter the number of classes and the length of queues.

SCHEDULER: It chooses the jobs from input queues on the bases of depending on classification. When a job has been selected it is subdivided into activities. Activities are identified by sequence numbers within a job. From this time the GCOS will work on activities. Another decision point may be the developing of priority strategy.

PERIPHERAL ALLOCATOR: It determines the requirements, evaluates the priorities and allocates the free peripheral units.

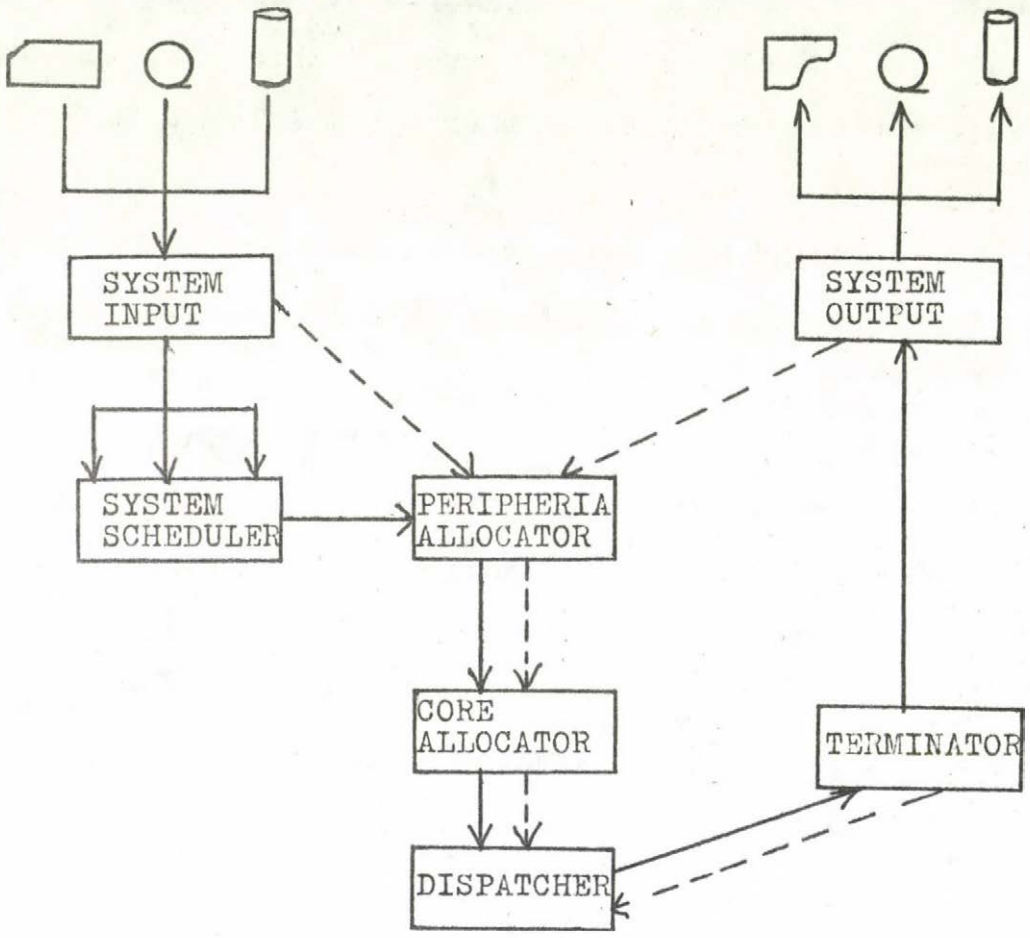
CORE ALLOCATOR: A privileged program with the highest priority. It handles and allocates the memory space. Also has a special strategy: first to compress then to spown if necessary.

DISPATCHER: It allocates the processor, handles the interrupts by 64 msec. Next decision point can be the choice of one or more methods from the processor allocating-techniques. It is a memory-resident program.

TERMINATOR: It releases the devices and accounts for the usages. It is a resident program too.

SYSTEM OUTPUT: It collects the outputs of activities which belong to one job and executes the output functions. It's operation is similar to system input.

The job-flows and the relationships of GCOS modules are shown in Figure 5.



——— job flow

- - - - - GCOS modules flow

Figure 5. Jobs and GCOS elements flows

The central part of the model is the simulation of GCOS because its modules reflect the functional operation of the hardware elements. The GCOS modules manage the hardware resources. The dynamic management is limited by the capacity of resources, by the service and working time, and by the requirements of jobs. The HwB-SIM follows the hardware utilization and the scheduling algorithm exactly. Figure 6. shows a general working schema used by the GCOS modules.

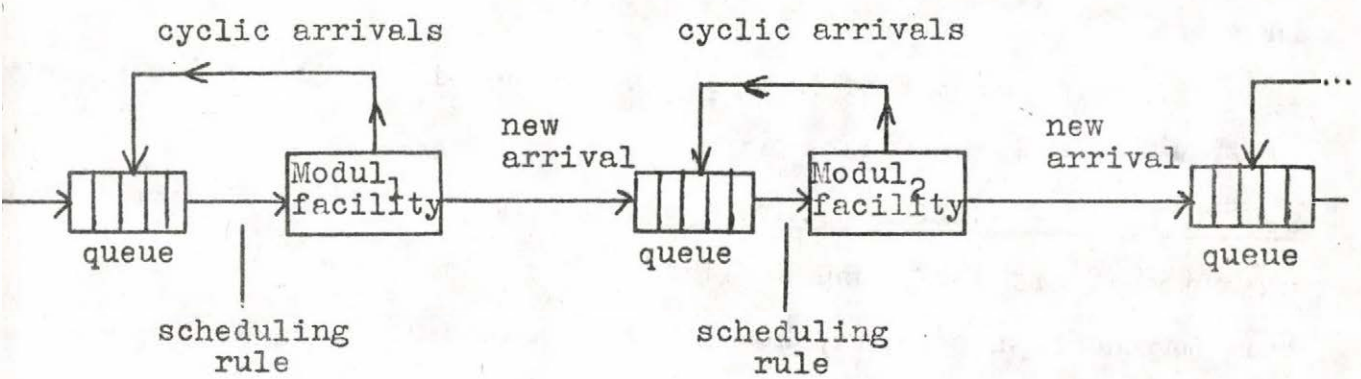


Figure 6. Queuing simulation

The queues are fictitious, really they reside on discs or in core. In the multiprogrammed environment the scheduling rule chooses the program to be executed. There are common characteristics in the scheduling algorithm of GCOS modules. 1/. The starting value of priority is determined by the user. The priority value may be in the interval 0-63. The default value is 5. The values from 56-63 are reserved for

the operating system.

2/. If two or more programs have the same priority value the scheduling sequence is FIFO.

3/. When the program arrives the resources are already reserved.

Thus one of the aims of GCOS is to minimize turnaround time. In every case of a failed attempt the priority is increased by 1. In addition to the common features every GCOS modul has special external priority strategy. E.g. in the case of printer without spooling mechanism the peripheral allocator increases the priority by 30.

Thus the main purpose of HwB-SIM is the setting of some parameter which are the following:

- SYSTEM INPUT - SCHEDULER interface: how many classes are set up and what are the resource limits of classes, how many members may a class contain, how it selects the job-mix that run at the same time.
- DISPATCHER strategy: there are four strategies at processor allocation:

  1/. "I/O priority". The processor allocation depends on the rate of channel time and processor time requirements. It prefers activities with the more channel time requirements. In each 192 msec it evaluates the queue again and the priority of failing requests is incremented by 1. Privileged programs are reevaluated more often.

  2/. Urgency throughput. The purpose of this strategy is minimizing the total job turnaround time. The priority /P/

will be:

$$P = \frac{\text{earlier priority} + 2}{8}$$

It is reevaluated in the case of I/O interrupt, fault, time-slice exhausted and when a new arrival occurs.

3/. Priority B class. At the startup parameters we may assign max. 3 jobs which will get processor more often than the others.

4/. Priority A class. At the startup parameters we may assign max. 5 jobs, and these jobs will be executed first and the others after them.

Model in SIMSCRIPT

The machine model of system is composed in SIMSCRIPT language, which allows a modular and hierarchic structure of formulising the problem. The jobs and activities are the temporary entities of the system. The processing requirements of jobs and activities are their attributes. The connections of jobs and activities are reflected by a set. The hardware resources are permanent entities and their attributes are the performance data and their status can be free or busy. The main memory is built up by the free and busy set of K words. The operation of peripheral subset is handled by an endogenous event routine. The modules of GCOS are operating in the frame of endogenous events. Every GCOS modul is defined as an event notice entity. The GCOS modul's information is stored in their attributes.

Those modules which require the service of other modules were defined as temporary entities, too. They are different from the normal programs because of their high priority, their fix program number and their attribute values. The queue which requires the same GCOS modul functions is handled like a priority ranked set by the HwB-SIM.
The event routines of HwB-SIM:

Type	Name	Function
exogenous	START	starting the run, temporary entities /jobs and activities/ values defining, starting condition defining
endogenous	INDUL	input device categoring, selecting the input job-stream
	GEIN	system input function
	SCHED	scheduler function
	PERI	peripheral allocator function
	CALC	core allocator function
	DISP	dispatcher function
	TERM	terminator function
	SYSO	system output function
	TAV	handles the peripheral subsystem

The STAT report generator forms and prints the results.
The motive power of HwB-SIM is the DISP. The execution of each program starts when the processor is allocating to it, and finishes when any of the following events occurs:

- fault /it is controlled by random number generator/
- I/O interrupt
- program termination
- time-slice exhausting /64 msec/.

Then the DISP reevaluates the waiting activities by changing the priorities and reallocates the CPU.

The causation chain of events - as discussed by Wyman /2/ - in HwB-SIM is the following:

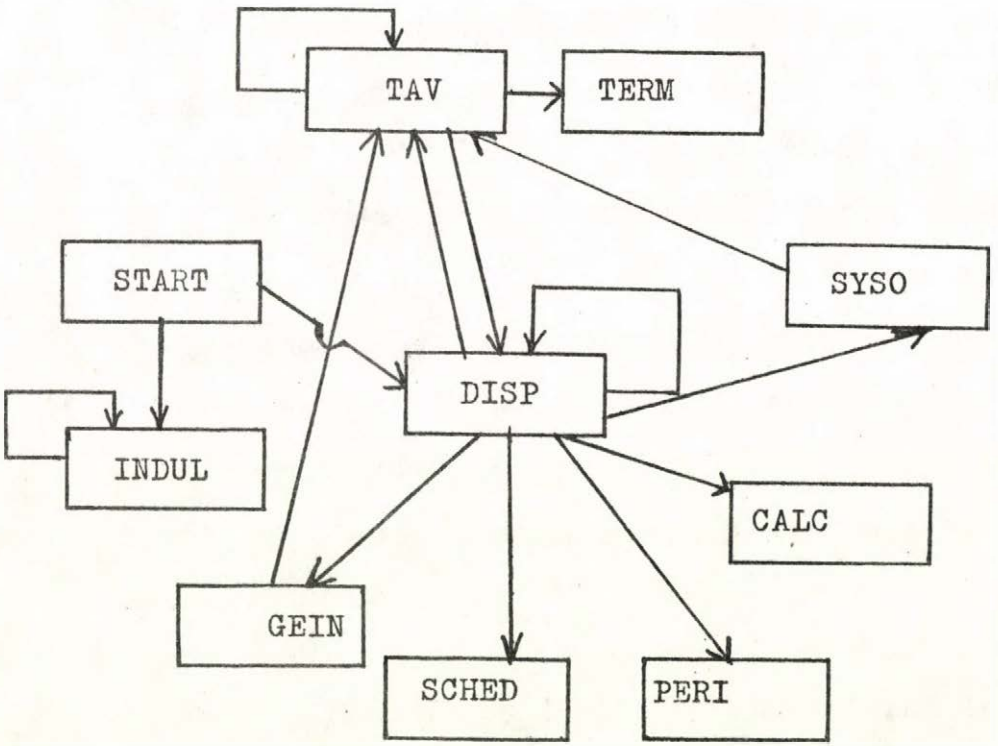


Figure 7. Causation of event routines

Methods of results evaluation, validation

We can analyze the system from different viewpoints:

- the requests of users
  - = to minimize the avarage turnaround time,
  - = to minimize the avarage delay of jobs.
- the requests of operators /maintaner/:
  - = to maximize the average utilization of devices,
  - = to minimize the idle time of units,
  - = to increase the throughput.

We have chosen our model to maximize the utilization of hardware units with a suitable response time. We can compute this from the output of HwB-SIM:

- avarage turnaround time
- queue statistics for peripheral devices
- processor utilizations
- core utilizations /memory map, number of compress e.t.c./.

After building up the model we have done systematical series of runs to show the logical error of modelling. First of all the independent parts of program were run alone in order to justify the results. In the case of stochastic models the results of one run are regarded as an observation of statistical indices which belong to the system operating descriptions. Because of random effects these values vary from run to run. We can get an n-element sample if we do n runs with the same starting conditions but with different random number series. We can make estimates based on this

sample and can determine the confidence interval of estimating.

Simulation characteristics

The model consists of 2000 source statement. The simulation time units is 10 miliseconds. As in SIMSCRIPT the model has unequal time increments. It means that the system is to be evaluated when an event occurs. The avarage run time of model is half an hour. Core requirement is 50 K words.
We had two alternatives of choosing simulation languages: GPSS and SIMSCRIPT. We had chosen the SIMSCRIPT because of modularity and algorithmic characteristics. From the point of view of programming efforts the SIMSCRIPT language has more advantages in the cases of high complexity. This is shown in Figure 8. The figure is based on our empirical observations.

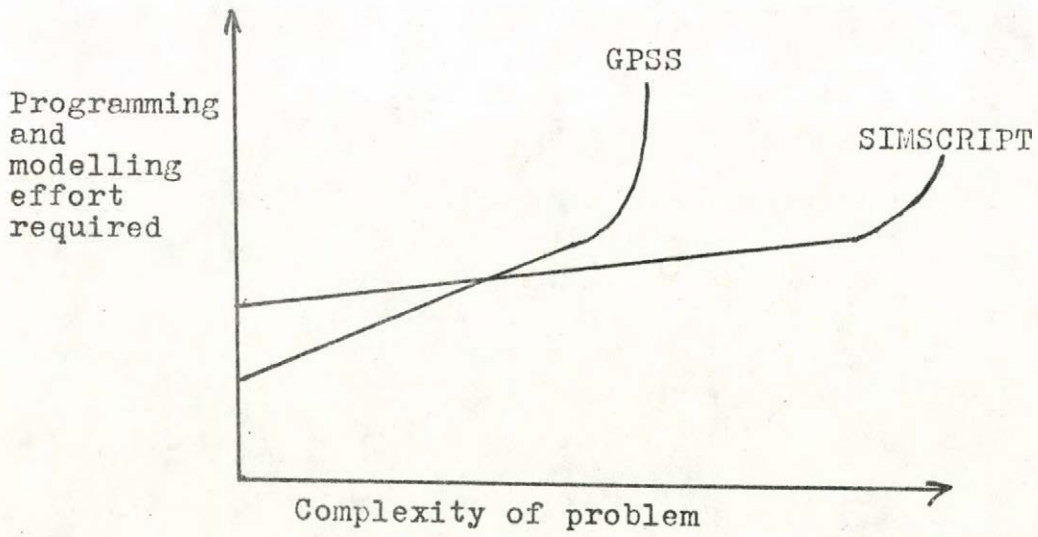


Figure 8. SIMSCRIPT and GPSS characteristics

Conclusion and developments

The conclusion of our operating model is that the computer system is very responsive to the different running strategies, e.g. for the dispatcher and scheduler algorithm modifications. Further development of the model seems to be necessary. We are going to introduce the real time environment and terminal-network system processing.

Bibliography

/1/ Dr. J. P. Buzen: Practical use of queuing network models 1978 Summer Scool on Computer Systems Performance Eval.

/2/ F. P. Wyman: Simulation modeling: A guide to Using SIMSCRIPT.
John Wiley and Sons, Inc. New York 1970.

PROBLEMSOLVING STRATEGIES

G. Dávid

Computer and Automation Institute
Hungarian Academy of Sciences

In logic-programming one of the most important problems is to find a suitable searching strategy. Most of the problem-solving systems are based on mechanized theorem proving techniques (e.g. resolution principle) and the search is controlled by a program. This program can be treated as "strategy". Our aim is to give a language, in which problemsolving strategies can be formulated.

Here a short introduction to the logic programming is described to illustrate the objects of a strategy and after that the main features of a general-purpose strategy-description language will be described.

Logic programming

In logic programming the programming language is a suitable mathematical logical language e.g. first-order logic (1,2) or many-sorted logic (Structure Logic SL in 3,4). In this language the user can write statements, representing facts, or knowledge. The chosen logic has deduction rules, by which from two statements a third one can be deduced i.e. if S_1 and S_2 are statements then a deduction rule D may produce a new one, say $D(S_1, S_2)$ and this derived statement (if existed) is a logical consequence of the original statements S_1 and S_2.

The most interesting decutions are those which can be mechanized. The resolution principle is such a mechanical deduction rule, and valid in both the first order and the structure logics, but here will be illustrated in the 0-order logic (i.e. logic without variables, functions, constants) for sake of simplicity. The resolution will be represented as an operation $o(S_1, S_2)$ but we prefer the infix notation, hence we shall use $S_1 \circ S_2$:

If given two statements A_1 and A_2 in the form of

$$A_1 : \quad \sim S_1 \vee S_2$$
$$A_2 : \quad S_1 \vee S_3$$

where $\sim$ and $\vee$ represent "not" and "or" then the statement

$$A_1 \circ A_2 : \quad S_2 \vee S_3$$

(the socalled resolvent) is a logical consequence of A_1 and A_2.

Having mechanical decution rules, and if given a set of statements

$$AX = \{S_i \mid i = 1,\ldots,n\}$$

then a new statement S can be deduced from AX if there exists a chain (a refutation)

$$A_1, A_2, A_3, \ldots, A_k = S$$

such that if for every $i(\leq k)$ A_i is either an element of AX, or has been deduced from A_j's $j \leq i$ by one of the possible deduction rules. The chain with the appropriate deduction rules is called the proof of S.

The resolution theorem says the if S can be proved them it also

can be proved such a way that only the resolution principle is used as deduction.

Hence a mechanical theorem proving system may work as follows: Starting from the elements of AX, produce all the possible resolvents (set AX^1) and check whether S is a consequence of one of those or not. If yes, stop, if not, continue with the produced set AX^1 as before (with AX), and produce AX^2, check and continue if necessary. This method is the direct proof of S. Instead of this we use indirect reasoning in the following manner. Let $\square$ be the empty statement (the nil).

Start from the negation of S, i.e. the indirect statement what is to be proved. Produce the set of resolvents with ~S and the element of AX,

$$AX_1 = \{ \sim S \circ A \mid A \in AX \}$$

Check, whether $\square$ is element of the AX_1 or not. If not, continue. The typical set is

$$AX_n = \{A' \circ A'' \mid A' \in AX, A \in AX_{n-1}\} \quad n > 1$$

If AX_n contains $\square$ then this means a contradiction hence we get an indirect proof of S. If the proof of S exists, this algorithms will find that. Figure 1 illustrates this proving technique, and the so-called "searching tree".

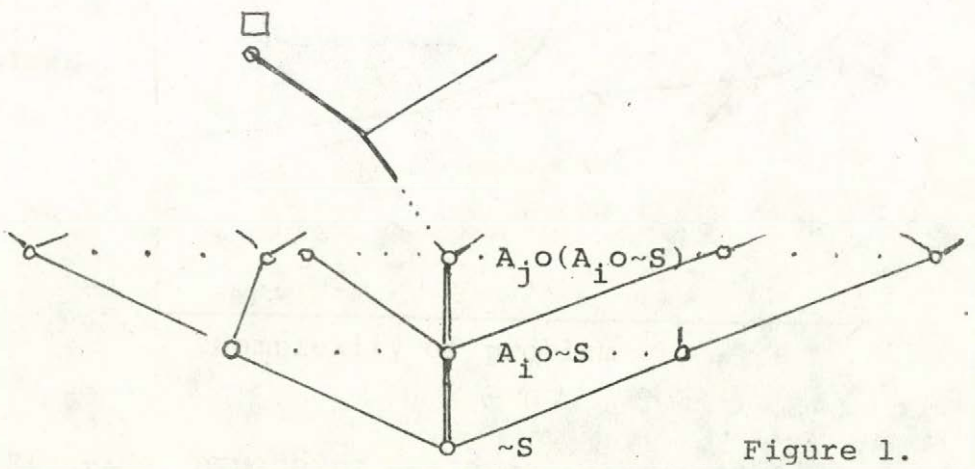


Figure 1.

It is easy to compute the number of new statements generated and one can imagine the "combinatorial explosure" which is the strongest limitation in present-days computer systems. Logic programming is important not only in Artificial Intelligence problems but in software specifications, in program synthesis (see 5), also, so we need to control the searching in the tree of Figure 1. Strategies do this control.

Strategies - both heuristical and mathematicaly well-founded ones - are important also in the hypothesis formation (6,7) robotics, etc. In the literature one can find a nice collection of strategies, even in the theorem proving, but also in the other branches of AI.

The control of the search in the tree means that in every step the strategy should

- select those statements which will be used to be resolved

- accept or reject the resolvent.

Moreover, a strategy produces a sequence of sets S_1, S_2, ..., S_1 - each set contains the accepted resolvents. The next set in this sequence is produced from the previous ones.

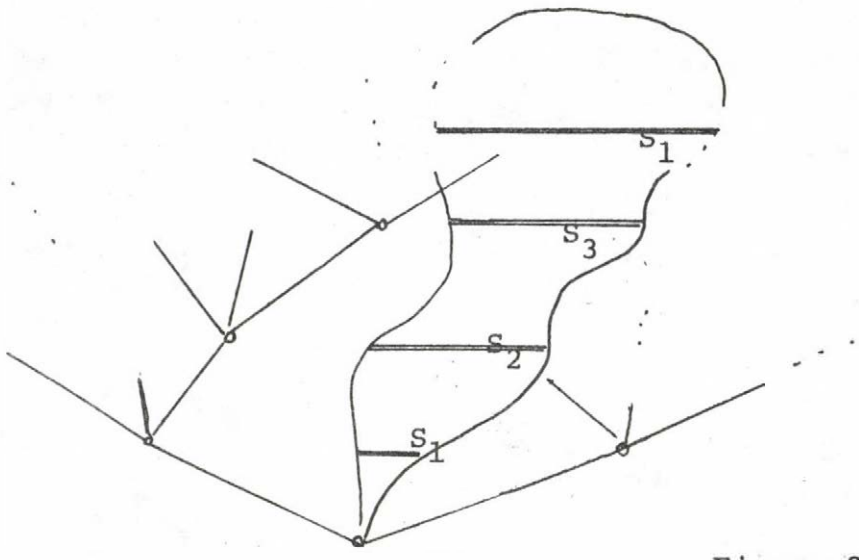


Figure 2.

Hence a strategy recursively defines the sets of resolvents. By now our goal more or less clear: to define a language in which one can describe various strategies.

Description of strategies

From the previous introduction we may see that the language for description of strategies should be based on set theory and recursively definiable sets. But to formulate the mathematical language of sets as a programming language we should extend it in the following ways:

- strategies deal with sets whose elements are statements. We want to define sets whose elements are defined by a logical language again but the models of this logic should be statements. So we need a language, on which statements on statements can be described.

- the strategy language should cover the mathematical language for sets (operations, etc.).

- the sequence of the sets, generated should be finite. But there is no criteria for finiteness. In every strategy the programmer should specify the failure condition (i.e. the condition, which - if it is fulfilled - stops the sequence) and that action if the strategy failed.

- a strategy may activate another one (or itself again), for example if the strategy had been failed.

In the sequel we proceed in this order and finally the general form of strategies will be given with simple examples.

Language of the statements on statements

As it was emphasized every strategy states something on sets, whose elements are logical statements. (Here we don't want to speak about the form of statements. They may be well-formed formulas, clauses, Horn clauses and such.)

We need a first-order logic language with

constant-symbols
variable-symbols
function-symbols (with arbitrary arity)
predicate-symbols
logical connectives $\vee$, $\wedge$, $\rightarrow$, $\sim$,
quantifiers $\exists$,$\forall$,
auxiliary symbols (,)

Constant symbols are either literals - i.e. the statements themselves - or a label of a statement in the form of

C: statement

Variable symbols are those which are not specified elsewhere. A special variable symbol is represented by

new Z

where Z is a set-symbol (see later).

In our language the set of function-symbols whould be specified as well as the predicate-symbols. They are not only symbols but executable functions or evaluatable predicates. For example, unary functions may be:

V(s) - the number of variables in the statement s;
C(s) - the number of constant symbols in the statement s;
N(s) - the degree of nesting of function in the statement s;

L(s) - the number of literals in s

etc.

They are integer-valued functions. Also we need statement-valued ones, for example

IMP(s_1, s_2) - has as value that s_i for $s_i \rightarrow s_j$ $j \neq i$, or nil if neither $s_1 \rightarrow s_2$ nor $s_2 \rightarrow s_1$ is valid

In the set of predicate symbols we have evaluatable predicates, having T (true) or F (false) as possible values.

Examples:

IMPLIES (s_1, s_2) - true, if $s_1 \rightarrow s_2$
EQU (s_1, s_2) - true, if $s_1 \leftrightarrow s_2$
EQUST (s_1, s_2) - true, if s_1 as string equals with s_2
ELEMENT (s_1, s_2) - true, if there exists a substitution τ of s_1 that $s_1\tau$ equals with a part of s_2 (as strings)

Well-formed formulas are composed by logical connectives, guantifiers and auxiliary symbols.
It should be noted, that although we had choosen a first-order language, and this language can be interpreted various ways, we gave an interpretation of them - and this interpretation is based on the domain of statements. As a programing language, every function and predicate symbol must be evaluated, hence implemented.

Language of sets

In this respect we accepted the mathematical notation for sets, with some extensions.
Sets are defined as logical statements (well-formed formulas)

of the language given previously. Sets are represented by bracketing, using { and } with appropriate restrictions. General form of the set-definition:

set-name = { element | statement }

The set-name is defined by the programmer, the element represents a typical element of the set, on which the statement should be true, or "element" represents the list of elements (separated by commas) and in this case the statement-part may be empty. Examples:

$A = \{ s_1 \circ s_2 \mid (\forall s_1 \in AX) \wedge (\forall s_2 \in step(n-1)) \}$

$B = \{ c_1, c_2, c_3, c_4 \mid (c_i \in B) \rightarrow N(c_i) > 2) \}$

$C = \{ s_3, s_4, s_5 \rightarrow s_6 \}$

(In order to explain the example above, we should remind the reader that o represents "resolvent", AX and step(n-1) are sets, the "$\in$" and ">" are infix symbols for the predicates "element-of", and "greater than", respectively, $N(c_i)$ represents the degree of nested functions in c_i, and $s_5 \rightarrow s_6$ is a statement again in the definition of C.)

New sets can be constructed by the operations over sets

$\cup$ (union)
$\cap$ (intersection)
$-$ (complement of)

Set-expressions are formed by this operations bracketing by "(" and ")". Set-assignments are in the form of

set-name = set-expression

Special functions of the language of statements on statements

can be defined by sets. An example is the new function, defined on sets and represented in the form of new "set-name" without bracketing. For every set, the writer of the strategy should specify the selection of an element. The new represents the selected statement or one of the selected statements. If the selection had not been specified then the selection is made on internal basis - the programmer hasn't any right to control this process. For every set to be controlled the programmer should define the operation new and activate that.

Definition of the selection: Let Z be a defined set, then one can define the selection by writing

$$\text{new } Z = \{ ST_Z \},$$

where ST_Z is a statement.

For example

$$\text{new } AX = \{ \forall s(s \in AX) \wedge (N(\text{new}(AX)) \geq N(s)) \}$$

The set-definition in the right-part selects a subset of the set AX, and new AX represent this subset. One could define finer selection also writing

$$\text{new new } AX$$

and defining again.

The new is always associated to the set in the definition. Hence if AXX is another set, and the set-assignment

$$AXX = AX$$

then the definition of newAX overwrites the definition of newAXX (if any) and the newAXX acts as newAX. More generaly, for set-assignments.

set = set-expression

then

new set = new set-expression

where the new "set-expression" is defined (by the statement ST_X for arbitrary set X) in the following way:

$$ST_{s_1 \cup s_2} = ST_{s_1} \vee ST_{s_2}$$

$$ST_{s_1 \cap s_2} = ST_{s_1} \wedge ST_{s_2}$$

$$ST_{\bar{s}_1} = \sim ST_{s_1}$$

With the already defined selection one can refer the selected statement by simply writing new Z for the set Z. Although the new Z defines a subset, and not a unique element, but the new Z serves one of them.

The action, performed by new Z may be represented as

Z = Z-{new Z}

i.e. the selected statement will be removed from the set Z.

Failure conditions and activate strategy

A strategy is succeeded if the $\square$ (nil) statement had been derived. To explain the failure of a strategy is not so simple. The programmer of the strategy should specify those conditions (that if the conditions hold them the strategy is qualified as failed) by

failure condition

"condition" is a statement in the logical language on statements, and a strategy can be activated by

on failure activate strategy;

In the condition one can state something on statements or on the proof. A proof is the actual value of a strategy.

We have seen that the strategy describes how the sequence of sets (of statements) will be generated. At every moment the strategy has as its value the generated sequence of sets, say

$$A_1, A_2, \ldots, A_k$$

They represents (derived) statements in the searching-tree, but A_k contains those which can be proved by a proof with length k. Hence they represent subsets of the first k layers in this tree. Two important consequences should be noted:

- the generated sets must be indexed by the length of the proof,

- the generated set sould not be empty.

From the second remark we state that a strategy is failed (regardless to the specified condition) if one element of the sequence of the sets is empty. From the first remark it follows that the generated sets will be indexed - we choose the parametrization form, assuming that the first parameter always represents the index. We do so, because strategy generaly depends on some other parameters also, and this fact should be reflected in the description.

In the failure - specification the condition may state something on the proof, for example the length n of the proof:

failure ((n ≥ 31) ∨ IMPLIES(A,B))

If the condition holds then the strategy may activate another one (or itself). This is specified by

on failure activate strategy,

where "activate strategy" is a set-name with actual parameters representing the strategy which generates it. The on failure specifications may be chained and the next one will be executed if the previous one is failed. One can activate strategy without on failure - in that case the strategy activates the other one and waits for its succes or failing.

Strategy as a programming language

Until now we described separate features of the strategies, in this section the programming language form will be treated. General form of a strategy:

strategy-name: initial specifications;
n-th step;
failure-part.

The three parts of the strategy should be separated by semicolon ;. Both initial specification and n-th step is composed from the followings:

set-definition,
set-assignments,
selection-strategy description,
activate strategy,
on-failure actions.

They will be interpreted sequentially.

Examples (S is the statement to be proved and AX is the initial set of facts).

strategy 1: step(o) = { ~S};

step(n) = {A o B | ∀A(A∈AX)∧∀B(B∈step (n-1))};

failure (n ≥ 31);

strategy 2: newAX = { ∀s((s∈AX)→(N(newAX)≥N(s)))},

level(1,C,~S) = {Co~S | C= newAX},

AXIOM(1) = AX;

end of initial definitions

level(n,A,B) = {AoB|(A=newAXIOM(n))∧(B∈level(n-1,C,D))},

AXIOM(n+1) = AX,

level(n+1, newAXIOM(n+1), AoB),

if the level (n,A,B) is successful then it turns to the next layer with the generated element.

on failure level (n, newAXIOM (n), B),

i.e. if failed then strategy tries to continue with the same B but with another "A", so it returns to the last decision in this layer, when the new was activated.

on failure level (n-1, newAXIOM(n-1), D),

if it was failed for every element of AXIOM(n) then it turns back to the previous level;

failure (n≥50) ∨ (n=0);

These strategies are well-known in the literature (bredth-first and depth-first strategies).

References

1 R. Kowalski: Predicate Logic as Programming Language
Proc. IFIP Congress 1974, North-Holland

2 D. Warren, L. Pereira: PROLOG-The language and its implementation compared with LISP
Proc. of ACM SIGART-SIGPLAN Conf. on "AI and Programming Languages", Rochester, N.Y. August 1977

3 G. Dávid: Structured Automatized Design of Microprograms, in Large Scale Integration
H.W. Lawson et al (eds.), North-Holland 1978

4 G. Dávid, S. Keresztély, I. Losonczi and A. Sárközy: Logic-Based Description of Microcomputers
MTA SzTAKI Tanulmányok 91 pp. 75-92, 1978

5 - : Microprogram Synthesis
MTA SzTAKI Tanulmányok 91 pp. 187-206, 1978

6 P. Hájek, T. Havranek: Mechanizing Hypothesis Formation
Springer-Verlag Universitex, 1978

7 D.B. Lenat: The Ubiquity of Discovery
Proc. of 5th IJCAI, Cambridge, 1977, pp. 1093-1105

DYNAMIC HASH-CODING : NEW SCHEMES FOR STRUCTURING DATA THE VOLUME OF WHICH IS ALLOWED TO GROW AND SHRINK BY LARGE FACTORS

Michel SCHOLL
IRIA - LABORIA
B.P. 105
78150 LE CHESNAY

New file organizations based on Hash-Coding recently appeared in the literature, which are suitable for data whose volume may vary rapidly. In the three schemes which have been independently proposed, rehashing is avoided, storage space is dynamically adjusted to the number of records actually stored and there are no overflow records. Two of these techniques employ an index to the data file. Retrieval is fast, however storage utilization is low (Ln2 $\simeq$ .69).

In order to increase storage utilization, we introduce below two schemes based on a similar idea and analyze their performance. Both techniques use an index of much smaller size. In both schemes, overflow records are accepted. The price which has to be paid for the improvement on storage utilization is a slight access cost degradation. For a storage utilization equal to .85, the expected number of accesses to secondary storage required to find a record (successful search) is less than 2 (provided the index is available in main storage).

1 - INTRODUCTION

Conventional Hash-coding used as a file access technique has the advantages of being simple and fast (at least if the distribution of the records over the available storage space is uniform) (see, for example, [1, Section 6, 4]. However, if the file grows by large factors, or if the record's distribution over the available storage space is not uniform, the number of overflow records may be large and therefore retrieval of a record may be significantly slowed down. On the contrary, if the file shrinks, storage space is underutilized. Such situations require the file's rehashing, which is costly, especially in a multi-user environment.

Recently, new file organizations based on Hash-coding appeared in the literature which avoid rehashing or any through reorganization of the file when the data volume grows or shrinks by large factors. In the three schemes which have been proposed under the names Dynamic Hashing [2], Extendible Hashing [3] and Virtual Hashing [4, 5], there are no overflow records. Assume the allocated secondary storage space is divided into buckets having a capacity of b records. When a record is to be inserted into a full bucket, the latter is split into two buckets among which the records are distributed[1]. The "hash" function, which locates a given record provided with a unique key, is dynamically modified and the allocated storage space is dynamically adjusted to the number of records actually stored in the file.

Dynamic Hashing [2] and Extendible Hashing [3] employ an index to the data file. By using the dynamic "hash" function, a bucket is associated with the given unique record's key and the bucket's location is identified by searching through the index. Once the bucket's address has been found, retrieval is fast : only one access to secondary storage is required, since there are no overflow records. If the file grows steadily, this index, initially available in core memory, will eventually be partly stored in secondary storage. This will slow down searching and updating. Extendible Hashing's index rapidly becomes prohibitively large, while in the case of Dynamic Hashing, the expected index size grows linearly as the number of records increases. However, Dynamic Hashing index has a tree structure which may render the process of searching an index

---

1) This splitting policy is used in connection with other file organizations, as IBM's Virtual Storage Access Method (VSAM) [6] and B-trees [1] .

entry slow if part of the index is stored in secondary storage. The two Virtual Hashing schemes proposed in [4, 5], respectively called VH_0 and VH_1 do not employ any index. Retrieval of a record requires only one access to secondary storage with VH_0. The price to be paid for this is a very low storage utilization, compared to the storage utilization provided by Dynamic Hashing and Extendible Hashing, which is in both cases approximatively equal to Ln2 [2, 3][2].

In order to prevent VH_0' storage utilization degradation, it is suggested in [5] that splitting of a bucket be deferred (i.e. overflow records are accepted). This led to the definition of VH_1 [5]. However the lower bound on storage utilization is still low and deletion of a record is a rather complicate operation when the file shrinks (this is not dramatic in applications, where the file tends to grow steadily, e.g. business applications).

This idea has also been independently studied in [2][3]. The resulting improvement is not significant and leads to more complicated searching and updating algorithms.

In this paper, we pursue the same goal : splitting of a bucket is deferred until a certain number of overflow records have been inserted. Retrieval of a record may then require more than one access to secondary storage. We present below two dynamic hash-coding schemes provided with an index and analyze their performance. Compared to the above schemes [2, 3], these new techniques provide a smaller index size and a higher storage utilization while the access cost degradation is not dramatic and the searching and updating algorithms are kept simple.

The first scheme to be studied is a variant of Larson's Dynamic Hashing [2]. In this technique, splitting of a bucket is deferred until $\beta \times b$ records have been addressed to this bucket, where b denotes the bucket's capacity

---

2) It is not surprising that both schemes provide the same storage utilization. Only the dynamic "hash" functions and therefore the index structures are different. One may easily show that a variant of VH_0 with index would not only give the same performance (Ln2), but also closely ressemble to Extendible Hashing [3, 4].

It is interesting to note that this value of Ln2 obtained in [2] and [3] by different approaches is precisely the value found in the case of two other file organizations : VSAM [7] and B-trees [8].

3) A similar idea has been proposed for indexed sequential files [7] which results in an important increase in storage utilization.

and β is greater than 1. Expected storage utilization and number of accesses to secondary storage required to find a record (successful search), provided the index is available in main storage, are analyzed as a function of β. Compared to Larson's scheme, the index size is decreased by a factor of 1/β. In order to get a significant storage utilization increase, β must be large enough ($\beta \geq 3$).

A different approach led to the definition of the second scheme proposed below. In this technique, splitting is performed every $\gamma \times b$ insertions, and the bucket to be split is not necessarily the one in which the last insertion occurred[4]. This scheme provides the same advantages as Dynamic Hashing with deferred splitting, i.e. increase in storage utilization without significant increase in the number of accesses to secondary storage. Moreover less storage space is required for the index, thus the file may support a larger number of records before the index overflows on secondary storage.

2 - DYNAMIC HASHING WITH DEFERRED SPLITTING

We first outline the main features of Larson's Dynamic Hashing scheme. The reader is referred to [2] where the file organization and the searching and updating algorithms are given in more details. A first modification to Larson's scheme is then presented and analyzed, in terms of storage utilization and number of accesses to secondary storage. Finally another change in the data file structure is suggested, which slightly complicates the file structure but results in a significant improvement of the performance. In both cases, the average index size is decreased by a factor of 1/β.

The number of records actually stored in the file at a given point in time is denoted by N. Each record, identified by a unique key, is associated with a bucket. For the sake of simplicity, records are assumed to be equal in length. This is not crucial (see [2]). Initially, secondary storage space is allocated for M buckets. All buckets are assumed to be equal in size (b). Throughout the paper, a storage management system is assumed from which buckets can be requested and to which free buckets can be returned. Such a storage management system is used, for example, in the data base management system INGRES [10].

---

4) A similar splitting policy is proposed in [9].

The expected secondary storage utilization, and the expected number of accesses to secondary storage when (successfully) searching a record are respectively denoted by α and T.

2.1 - Dynamic Hashing [2]

An initial hashing function H_0 distributes the records among M initial index entries. Each index entry (node) contains a pointer to a bucket in the file. When trying to insert a record into a bucket which already contains b records, a new bucket is allocated, the b + 1 records are distributed among the two buckets, and the index is updated : the initial index entry becomes the root of a tree, two nodes are linked to the root, the left son pointing, say towards the original bucket, the right son pointing towards the new bucket. Sooner or later one or the other of the two buckets will in turn become full and will be split into two buckets, etc... The index grows with time to a forest of M trees. Assume the number of records actually stored decreases. If the sum of the number of records in two brother buckets becomes less than b records, the two brother buckets are merged into one, the other one being freed and the index is updated. A straightforward organization of the index forest [2] is depicted in Figure 1 : each index entry is either an internal node (TAG = 0), which contains pointers to its father and sons, or an external node (TAG = 1) which contains, besides the pointer toits father, a pointer to a bucket in the data file (BKT) and the number of records actually stored in this bucket (RCDS).

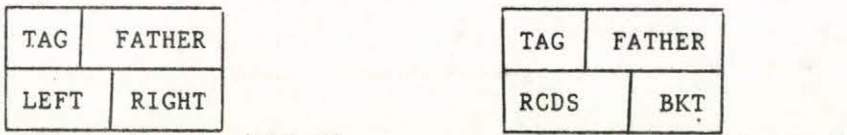


Figure 1 : Index entry

This simple tree structure leads to heavy storage requirements. Better ways of representing trees are discussed in [1].

To find or insert a record with key K_i, the root of the corresponding search tree is first located by computing $H_0(K_i)$. Then the tree is scanned down until an external node is reached. (This node contains the pointer to the bucket in which the record is to be stored). The problem now is to associate with each unique key K_i, a unique path when scanning down the tree. This is done, by

using a second function $B(K_i)$. It is suggested in [2] that B be implemented by means of a pseudo random number generator RAND which returns 0 ar 1 with probability .5 when called. The pseudo-random 0 - 1 sequence obtained by successive calls of RAND uniquely determines the path through the corresponding tree whose root is given by $H_0(K_i)$). More precisely, if the root node has a tag equal to 1 (external node) then it contains the address of the bucket in which the record is to be found. Otherwise, compute $H_1(K_i)$ where H_1 is anaother hashing function independent of H_0. $H_1(K_i)$ is the seed for the generator RAND. A first call of RAND indicates whether the left son or the right son of the root has to be chosen. The chosen son is either an external node, or an internal node. In the latter case, call again RAND, etc... until an external node is eventually reached, which node contains a pointer to the bucket in which the record is to be stored.

2.2 - Dynamic Hashing with deferred splitting

Assume now that splitting of any bucket is deferred until $\beta \times b$ records ($\beta > .1$) have been addressed to this bucket. First, consider the case $\beta \leq 2$. When trying to insert a record into the "primary" bucket i which already contains b records a new "overflow" bucket is allocated (bucket k) and chained to bucket i. The new record is inserted into this overflow bucket. Any other record candidate to be inserted into bucket i, will be inserted into the overflow bucket, until the latter contains $(\beta - 1)b$ records (see Figure 2a). This "chaining with separate lists" collision resolution method is used in connection with several applications, among others INGRES [10]. When trying to insert a record into a bucket whose overflow bucket already contains $(\beta - 1)b$ records, then splitting occurs in exactly the same fashion as in Larson's scheme [2]. A nex bucket say j is allocated, the b records are distributed among bucket i and bucket j. Bucket k is freed (Figure 2b) and the index which has exactly the same structure as in [2] is updated. Searching and updating algorithms are omitted since they are much the same as in Dynamic Hashing. For further details, the interested reader is referred to [2].

(a) (b) just after the split. k is freed

Figure 2 : Dynamic Hashingwith deferred Splitting

If β is greater than 2, when the first overflow bucket is full, another overflow bucket is allocated and chained to the first one, a.s.o., until βb records have been inserted. For all values of β, when splitting occurs, the index is updated, i.e. the former external node (of level r in the tree) becomes an internal node pointing towards two external nodes (its sons) of level r + 1 in the tree, each of them pointing towards a chain of one or more buckets containing the records. Then the βb + 1 records are distributed ((k + 1)st call of RAND) among 1) the original chain of buckets, and 2) a new chain of buckets (one or more new buckets are allocated). Possibly one or more buckets which belonged to the original chain are freed.

Following Larson's analysis, we derive below the expected storage utilization α and the expected number of accesses T to secondary storage, required to find a record actually stored in the file (successful search), provided that the index is available in main memory.

Let us consider first the index size. When the number of records N is large, it was shown in [2] , because of the close connection with tries [1], that the expected number of internal and external nodes are given by the following excellent asymptotic approximations, in the case of Dynamic Hashing :

$$E(e - nodes) \approx \frac{N}{bLn2} - M \qquad (1)$$

$$E(e - nodes) \approx \frac{N}{bLn2}$$

Obviously, the same argument applies to Dynamic Hashing with Deferred Splitting and we have :

$$E(e - nodes) \approx \frac{N}{\beta bLn2} - M \qquad (2)$$

$$E(e - nodes) \approx \frac{N}{\beta bLn2}$$

Therefore the index size is approximately decreased by a factor of $1/\beta$.

Denote by E(B) and $E(B_{ov})$ respectively the average number of "primary" buckets and the average number of overflow buckets in the file. Then we have :

$$\alpha = \frac{N/b}{E(B) + E(B_{ov})} \tag{3}$$

The number of "primary" buckets allocated is equal to the number of external nodes less the number of (non-allocated) empty buckets [2]. The number of empty buckets is negligeable when $\beta b \geq 9$ (see [2, Eq. (21)]). Therefore E(B) is given by the following asymptotic approximation :

$$E(B) = \frac{N}{\beta B \ln 2} \tag{4}$$

In order to find an asymptotic approximation of $E(B_{ov})$, we assume first that β is integer (2). Assume n records hash to tree t, t = 1, 2,...,M.

The average number of overflow buckets in tree t is given by :

$$E(b_{ov}) = \sum_{r>0} 2^r \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} q_r(i) \tag{5}$$

where $q_r(i)$ denotes the probability that at least i records went through a given node of level r in tree t (i among the n records produce the same binary sequence of length r, when successively calling RAND r times), given that more than βb records went through its father.

$q_r(i)$ is given by [2] :

$$q_r(i) = \sum_{k=b+1}^{n} \binom{n}{k} (2^{-r+1})^k (1 - 2^{-r+1})^{n-k} \binom{k}{i} 2^{-k}$$

$$E(b_{ov}) \triangleq \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \sum_{r>0} 2^r \sum_{k=\beta b+1}^{n} \binom{k}{i}\binom{n}{k} (2^{-r+1})^k (1 - 2^{-r+1})^{n-k}$$

$$E(b_{ov}) = \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} Q_i \tag{6}$$

where

$$Q_i = \sum_{k=\beta b+1}^{n} \binom{k}{i}\binom{n}{k} 2^{-k+1} \sum_{r>0} 2^{-r(k-1)} (1 - 2^{-r})^{n-k} \tag{7}$$

Applying Euler Mac-Laurin Sum formula to the last sum denoted by a_k, we obtain the following very good approximation :

$$a_k = \frac{k-2}{n-k+1}\, a_{k-1}$$

$$a_k = \frac{(k-2)!}{(n-k+1)(n-k+2)\ldots(n-2)}\, a_2$$

Applying again Euler Mac Laurin Sum formula to a_2 reduces to the following expression :

$$a_2 = 1/(n-1)Ln2$$

Then, we have :

$$Q_i = \frac{n}{Ln2} \sum_{k=\beta b+1}^{n} \binom{k}{i} \frac{2^{-k+1}}{k(k-1)}$$

$$Q_1 = \frac{n}{Ln2} \sum_{k=\beta b}^{n-1} \frac{1}{k2^k}$$

$$Q_1 \simeq \frac{n}{Ln2} \sum_{k=\beta b}^{\infty} \frac{1}{k2^k} < \frac{1}{\beta b 2^{\beta b-1}}$$

Therefore, for practical values of b, Q_1 may be neglected, and for $i \geq 2$:

$$Q_i = \frac{n}{Ln2}\left[\frac{2^{-i+1}}{i(i-1)} \sum_{0}^{n-i} \binom{k+i-2}{i-2} 2^{-k} - \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}\right]$$

The first sum into brackets converges very rapidly (n-i is large) to 2^{i-1}. Q_i is thus reduced to the following expression :

$$Q_i = \frac{n}{Ln2}\left[\frac{1}{i(i-1)} - \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}\right] \tag{8}$$

Substituting Eq. (8) into Eq. (6) and cleaning up reduces to :

$$E(b_{ov}) = \frac{n}{bLn2}\left[H_\beta - 1 - b \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}\right] \tag{9}$$

where H_β denotes the harmonic series of order β.

Combining Eqs. (3), (4) and (9) gives the following asymptotic approximation for the expected storage utilization (β integer ≥ 2) :

$$\alpha = \frac{\text{Ln}2}{\frac{1}{\beta} + H_\beta - 1 - b\,A(b,\beta)} \qquad (10)$$

where $$A \triangleq \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \sum_{k=i}^{b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}$$

The sum A has us stumped. Numerical calculations of A(b, β) showed that for $\beta \geq 3$, as b increases, bA(b, β) is very rapidly independent of b ($b \geq 10$).

If $\beta \leq 2$, Eq. (6) is replaced by :

$$E(b_{ov}) = \sum_{i=b+1}^{\beta b} Q_i$$

$$E(b_{ov}) = \frac{n}{\text{Ln}2}\left[1 - \frac{1}{\beta} - b \sum_{i=b+1}^{\beta b} \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}\right] \qquad (11)$$

Combining Eqs. (3), (4) and (11) gives the following expression :

$$\alpha = \frac{\text{Ln}2}{1 - b\,B(b,\beta)} \qquad (12)$$

where $$B(b,\beta) = \sum_{i=b+1}^{\beta b} \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i}$$

Observe that Eqs. (10) and (12) give the same expression for $\beta = 2$. Table 1 illustrates the behavior of α as b increases up to 2. Clearly, the improvement du to deferred splitting is not dramatic, espacially if b is large.

β	α
1	.693
1.3	.696
1.5	.702
1.8	.730
1.9	.744
2	.762

b = 10

β	α
1	.693
1.3	.693
1.5	.695
1.8	.711
1.9	.723
2	.740

b = 20

β	α
1	.693
1.3	.693
1.5	.693
1.8	.699
1.9	.706
2	.722

b = 50

Table 1 : Increase in storage utilization due to deferred splitting ($\beta \leq 2$).

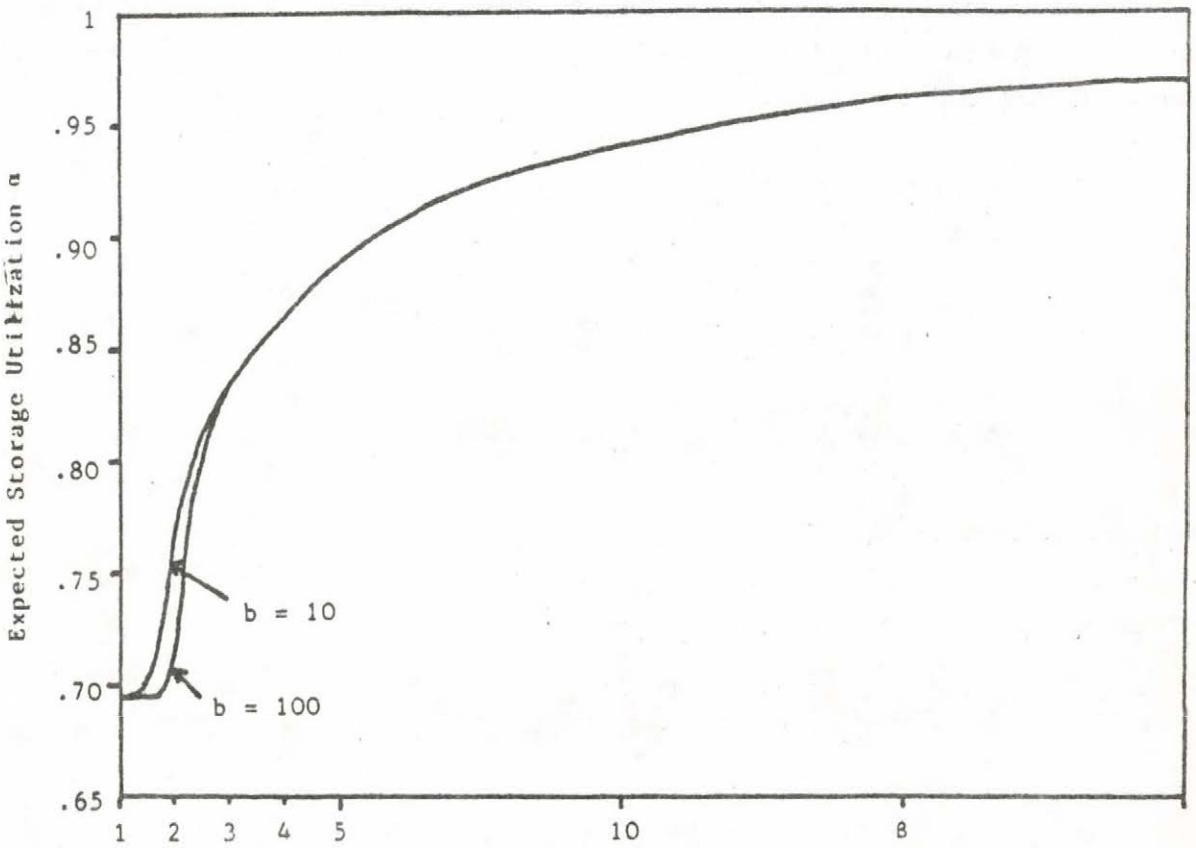


Figure 3 : Dynamic Hashing with deferred splitting : α vs β.

However, if β is large ($\beta \geq 3$), the increase in storage utilization is significant ($\beta = 3$, $\alpha = .836$; $\beta = 5$, $\alpha = .886$). This is illustrated in Figure 3 where α is plotted versus β.

Our last task is to find the expected number of accesses to secondary storage, T, required to find a record. Consider first, the case of β integer ≥ 2. Assume n records hash to tree t.

$$T = 1 + \frac{1}{n} \sum_{r>0} 2^r \sum_{j=1}^{\beta-1} \sum_{i=jb+1}^{(j+1)b} [b + 2b + \ldots + (j-1)b + j\,(i - jb)]\, q_r(i)$$

$$T = 1 + \frac{1}{n} \sum_{r>0} 2^r \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \left[i - \frac{(j+1)b}{2}\right] q_r(i) \tag{13}$$

$$T = 1 + \frac{1}{n} \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} Q_i \left[i - \frac{(j+1)b}{2}\right] \tag{14}$$

where Q_i is defined in Eq. (7).

Substituting Eq. (8) into Eq. (14), we get :

$$T = 1 + \frac{1}{Ln2} \left[\sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \frac{i-(j+1)b/2}{i(i-1)} - E(b, \beta) \right]$$

where

$$E(b, \beta) \triangleq \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} [i - (j+1)b/2] \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i} \tag{15}$$

We finally obtain the following expression for T :

$$T = 1 + \frac{1}{Ln2} \left\{ \sum_{j=1}^{\beta-1} j \sum_{i=jb+1}^{(j+1)b} \frac{1}{i-1} - \frac{\beta-1}{2} - E(b, \beta) \right\} \tag{16}$$

where $E(b, \beta)$ is given by Eq. (15). As β increases the first sum into brackets converges to $\beta - 2$ ($\beta \geq 9$). No significantly simpler expression of $E(b, \beta)$ has been found. Numerical calculations of the latter expression for various values of b showed, as for $bA(b, \beta)$ (see Eq. (10)), the independence of $E(b, \beta)$ with respect to b.

If $\beta \leq 2$, T is given by :

$$T = 1 + \frac{1}{n} \sum_{r>0} 2^r \sum_{i=b+1}^{\beta b} (i - b) q_r(i)$$

$$T = 1 + \frac{1}{n} \sum_{i=b+1}^{\beta b} (i - b) Q_i \qquad (17)$$

Substituting Eq. (8) into Eq. (17) gives :

$$T = 1 + \frac{1}{Ln2} \left\{ \sum_{i=b+1}^{\beta b} \frac{i-b}{i(i-1)} - \sum_{i=b+1}^{\beta b} (i - b) \sum_{k=i}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i} \right\} \qquad (18)$$

Cleaning up Eq. (18) reduces to the following expression :

$$T = 1 + \frac{1}{Ln2} \left\{ 1 - \frac{1}{\beta} - \sum_{i=0}^{b-1} (b - 1) \sum_{k=b+1}^{\beta b} \frac{2^{-k+1}}{k(k-1)} \binom{k}{i} \right\} \qquad (19)$$

One may easily verify that Eq. (16) is reduced to Eq. (19), if $\beta = 2$.

β	1.3	1.5	1.6	1.8	2	3	4	5	7	10	20
T	1.06	1.13	1.16	1.23	1.30	1.64	1.99	2.34	3.06	4.12	8.21

Table 2 : Deferred Splitting : Exprected Number of Accesses to Secondary Storage T

The access cost degradation is not large for small values of β, i.e. $\beta \leqslant 5$ (see Table 2).

In conclusion, Deferred Splitting significantly increases storage utilization if $\beta \geqslant 3$, i.e. if we let any bucket overflow 2 times its capacity before splitting it. Even though β is large, the value of T is reasonably low e.g. if $\beta = 3$, then $T = 1.64$). Figure 4 illustrates more precisely the trade-off between fast retrieval and high storage utilization. Each point of the curve is labelled by the corresponding value of β.

From Figure 4, it is clear that Deferred Splitting definitively improves Dynamic Hashing global performance. Indeed if $\beta = 4$, for example, the index size is divided by 4, the expected storage utilization is equal to .865, while on the average only 2 accesses to secondary storage are required to find a record (once the corresponding buckets chain has been located).

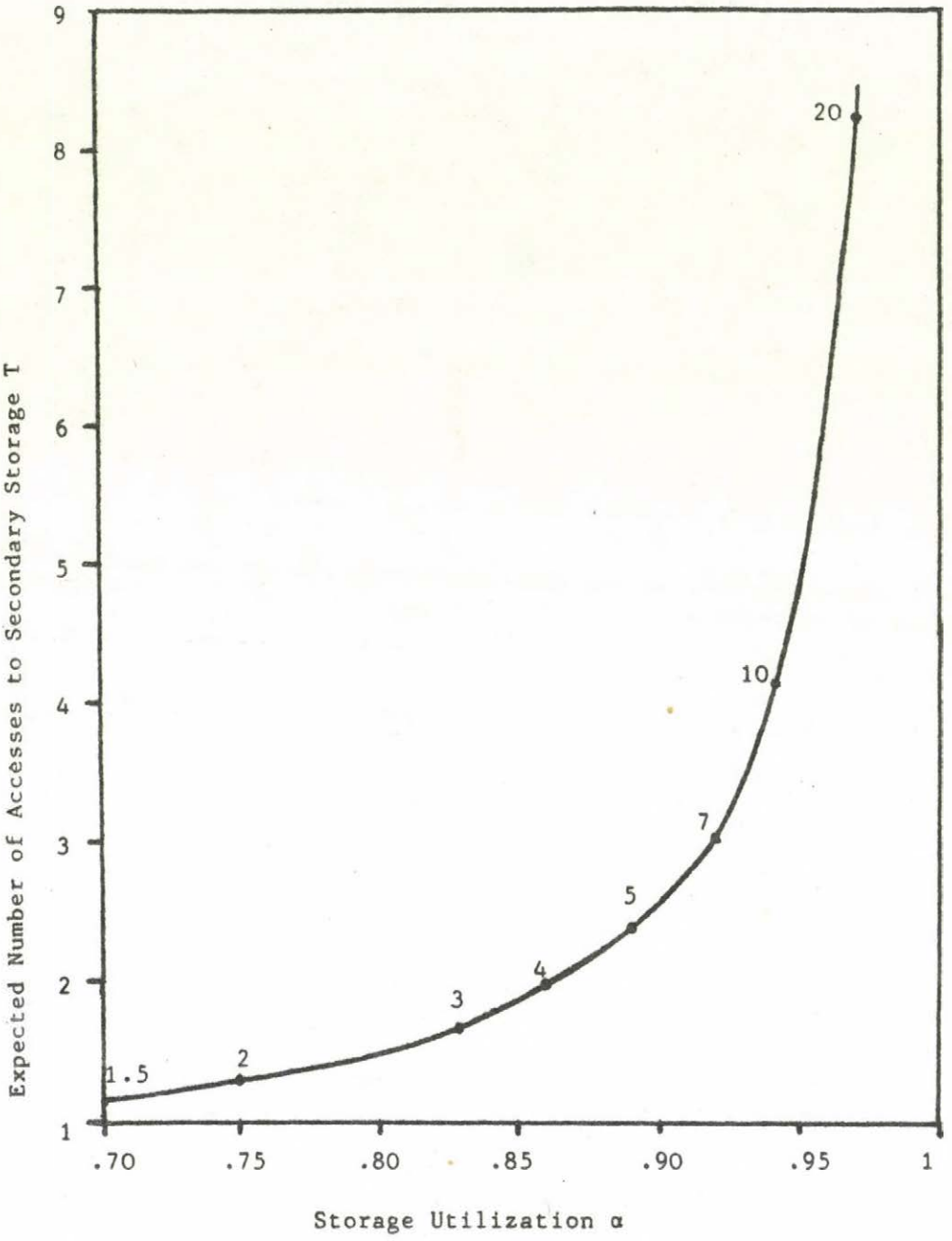
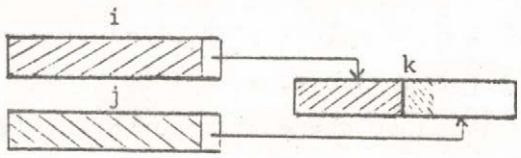


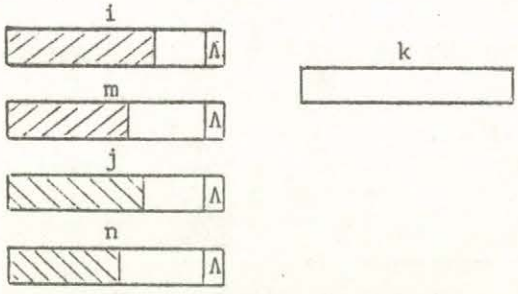
Figure 4 : Dynamic Hashing with Deferred Splitting : T vs α

2.3 - A Variant in which overflow buckets are shared among several primary buckets

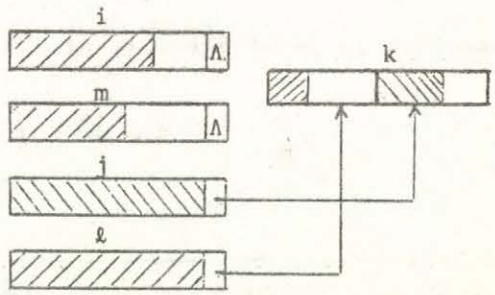
In order to give an idea of how the scheme works, the following example is taken. We choose the combination b = 10, β = 1.5, and assume as in Section 2.2, that splitting occurs when 15 records have been inserted (10 in the primary bucket i, and 5 in its overflow bucket). Assume, meanwhile, another primary bucket j has overflowed. Instead of allocating a separate overflow bucket to bucket j, we allocate the second half of bucket k, as an overflow area for bucket j (see Figure 5a). Eventually, either bucket i or bucket j will first be splitted say bucket i, and 15 records will be distributed among the initial bucket (i) and a new allocated bucket m. One half of the overflow bucket k is thus freed. Now one of the two following events may happen : 1) either (Figure 5b), bucket j is split, before any other primary bucket in the file needs the overflow space available in bucket k, and then (after the split of j) bucket k is freed, 2) or (Figure 5c) before j's split, bucket ℓ needs overflow space and is thus linked to the first half of bucket k.



(a) before splitting bucket i



(b) after the split of bucket j, k is freed



(c) k is not freed

Figure 5

Up to $1/(\beta - 1)$ primary buckets may share the same overflow bucket[5], e.g. if $\beta = 1.25$, one overflow bucket may be shared among up to 4 primary buckets, and 2 bits more are needed to identify (in the primary bucket), which part of the overflow bucket "belongs" to the given primary bucket. Following Section 2.2's analysis, we show below for $\beta \leq 2$, that storage utilization is considerably improved to the expense of increased complexity in buckets management.

The index size and the number of primary buckets are unchanged by the above modification, while the average number of overflow buckets is reduced by a factor of $\beta - 1$. Then from Eqs. (11) and (12), we obtain :

$$E(b_{ov}) = \left\lceil (\beta - 1) \frac{n}{Ln2} \left[1 - \frac{1}{\beta} - bB(b, \beta) \right] \right\rceil$$

The expected storage utilization is then given by the following asymptotic approximation :

$$(20) \qquad \alpha = \frac{\beta Ln2}{1 + (\beta - 1)^2 - \beta b(\beta - 1) B(b, \beta)}$$

In the range $1 \leq \beta \leq 2$, $\beta b(\beta - 1) B(b, \beta)$ is very small as shown in Figure 6. As b increases, the expression of α rapidly converges to the following (lower bound) expression (represented by the dotted curve in Figure 6) :

$$(21) \qquad \alpha = \frac{\beta Ln2}{1 + (\beta - 1)^2}$$

The maximum value of α occurs at $\beta = \sqrt{2}$:

$$(22) \qquad \alpha_{max} = \frac{(\sqrt{2} + 1)}{2} Ln2 = .837$$

---

5) If $\beta = 5/3$, then three primary buckets share two overflow buckets, i.e. one primary will have its overflow records split among two separate overflow buckets. This slows down retrieval and renders the algorithms more complex. Therefore β should be chosen less or equal to 1.5.

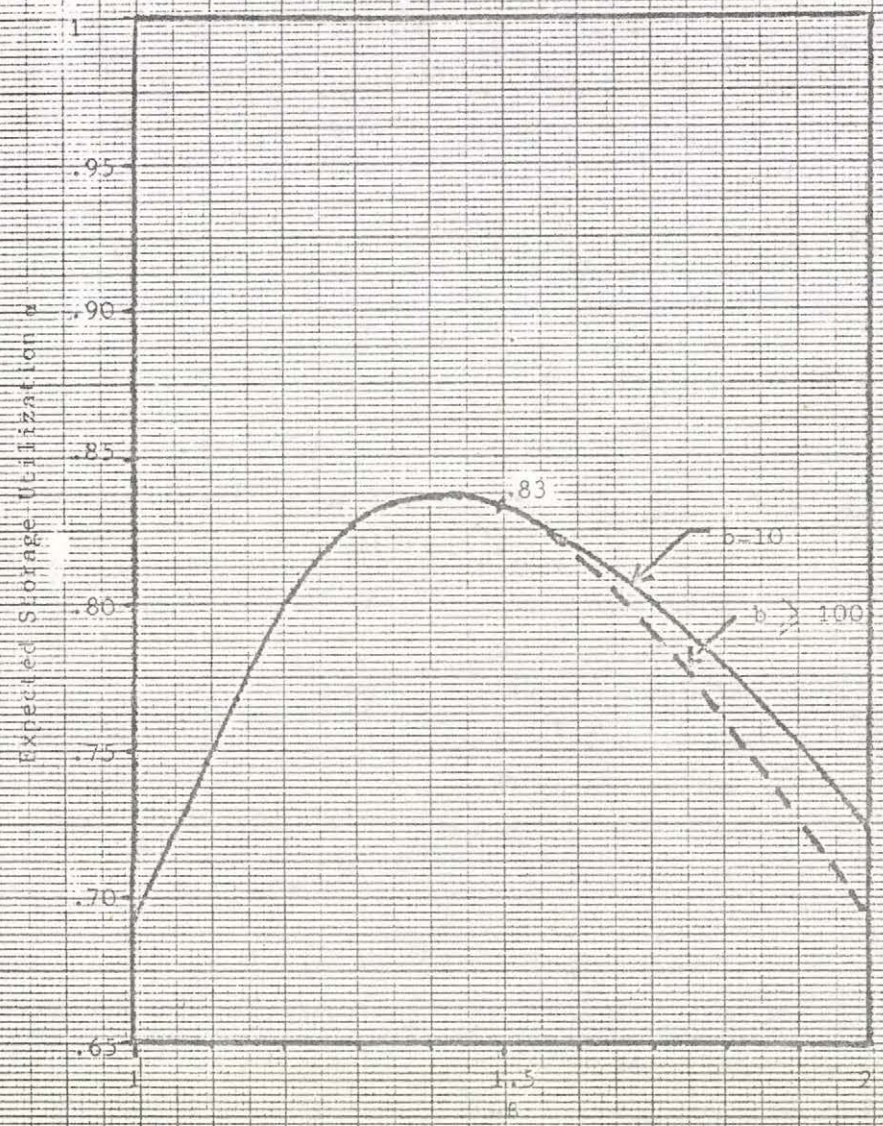


Figure 6 : Dynamic Hashing with Deferred Splitting (shared overflow buckets) : a vs s

For practical reasons β must be less or equal to 1.5 (see footnote 5) and such that $1/\beta-1$ is an integer. If we choose $\beta = 1.5$, then $\alpha = .83$ which is pretty close from the maximum value (see Figure 6 and Eq. (22)).

Observe that the expected number of accesses to secondary storage is the same as in Section 2.2 (Eq. (19), Table 2). Provided the index is available in main memory, one or at most two accesses are needed to find any record in the file. As a matter of fact, if $\beta = 1.5$ (i.e. for $\alpha = .83$), then $T = 1.13$, while in Section 2.2, if $\beta = .83$, then $T = 1.64$ (see Figure 4), and in the worst case the number of accesses is equal to 3. However, the scheme studied in Section 2.2 is more simple to implement than the present one.

3 - LINEAR SPLITTING

This technique too, necessitates maintenance of an index, each index entry pointing to a bucket of capacity b records. The tree structure of Dynamic Hashing's index may lead to heavy storage requirements (see Section 2). Besides, tree searching may considerably slow down retrieval of an index entry if part of the index is stored in secondary storage. One of the advantages of the following file structure is to considerably improve the index performance (less storage, easier search through the index).

3.1 - File structure

An initial function H_0 associates to the record's key K, one among M linear lists, say L. Another hash function H_r, independent of H_0, associates to K a unique entry in the Lth table (see Figure 7). This entry contains a pointer BKT to the secondary storage bucket where the record is to be stored : BKT = L(H(K)). H_r is dynamically changing according to the subfile size, i.e. according to the number of records which hash to list L ($L = H_0(K_i)$).

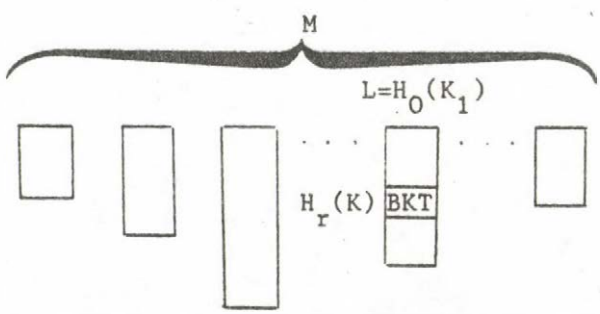


Figure 7 : Linear Splitting : Index Structure

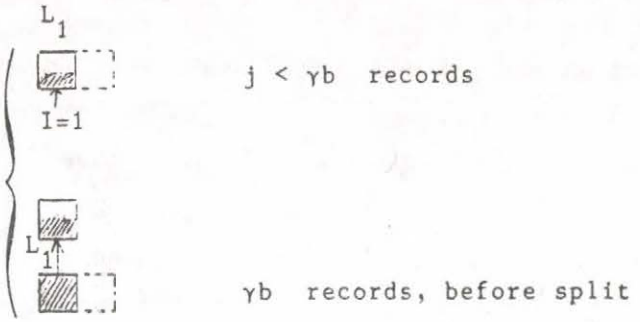


Fig. 8a
(r=0)

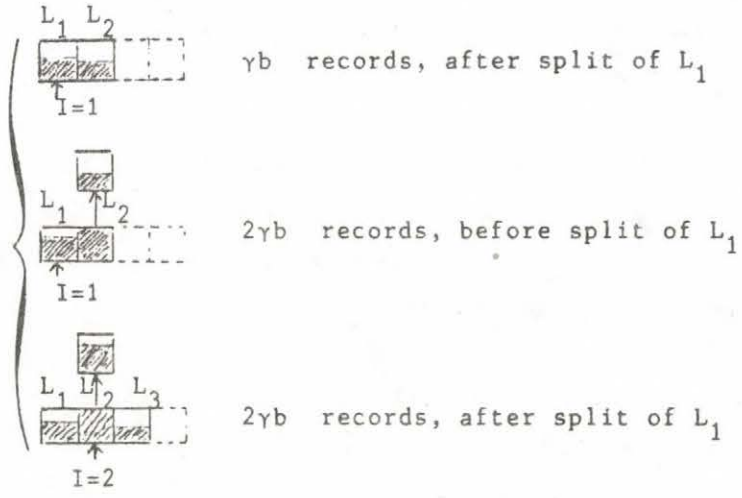
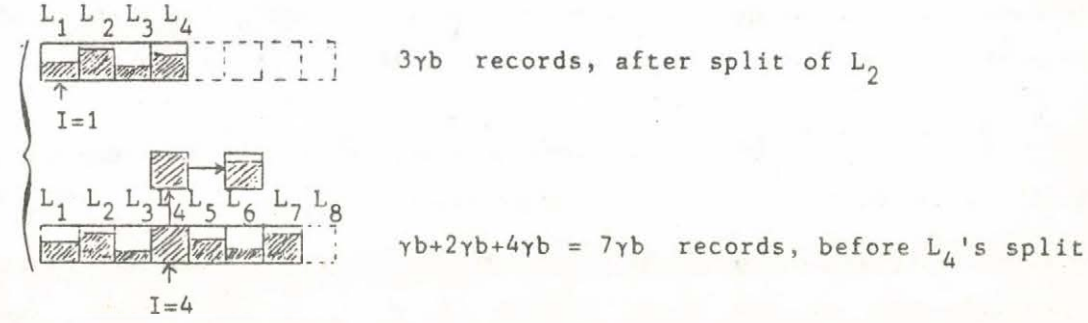
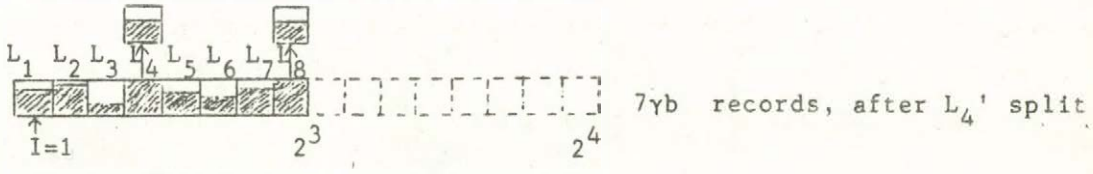


Fig. 8b
(r=1)



r=2

fig. 8c



r=3

Figure 8 : Linear splitting, Bucket's allocation in subfile L, L=1,...,M, (γ > 1)

First starting from an empty file, let us see how to insert N records. The initial index is composed of M cells (each list has only one cell), M buckets being initially allocated to the file. After γb records have been inserted in the Lth bucket denoted by L (γb records hashed to cell L) bucket L_1 is split into two buckets, the records being distributed among L and a new bucket L_2 (see Figure 8b), whose address is contained in a new cell L(2) appended to the Lth list (initially composed of one cell : L(1)). The index list L contains now 2 cells. After b new insertions, L_1 is split once more into two buckets, L_1 and L_3. Bucket L_3's address is contained in L(3) appended to the Lth index list. After b new insertions, L_2 is in turn split into two buckets, L_2 and L_4. The next buckets to be split are then L_1, L_2, L_3, L_4, L_1, L_2, L_3, L_4,...,L_8, L_1 etc... (Figure 8c). Eventually, as the file grows, any bucket may overflow before it is in turn split. Then, overflow records are stored into one more overflow buckets as in Section 2.2. When splitting occurs, one or more new buckets are allocated, while one or more buckets (in the initial chain) are possibly freed.

When the Lth list reaches 2^r cells (2^r buckets have been allocated for the records hashing to the Lth subfile), the next bucket to be split is L_1 ; when the Lth list has $2^r + I - 1$ cells, the next bucket to be split is L_I, $I = 1,...,2^r$, etc... ; when $I = 2^r$, the Lth list contains 2^{r+1} cells (Figure 9). Two supplementary cells are needed for each list, one containing the value of I, the other one indicating the "level" of the list. Each time, a split occurs, the pointer I is incremented by 1 (as well as r if $I > 2^r$).

Suppose the file steadily shrinks. After b deletions in the Lth subfile, two buckets are merged, respectively L_{I-1} and L_{2^r+I-1} (the latter is the most recently allocated bucket), the records being gathered into bucket L_{I-1} (and possibly into one or more overflow buckets). L_{2^r+I-1} and its overflow buckets are freed, the last index cell $L(2^r + I - 1)$ is deleted and I is decremented by 1. A third supplementary cell is needed for each list, which contains an integer n in the range [0, γb]. Each time an insertion (deletion) occurs n is incremented (decremented) by 1. When n = γb (n = 0), a split (merge) occurs. The insertion and deletion algorithms are summarized below.

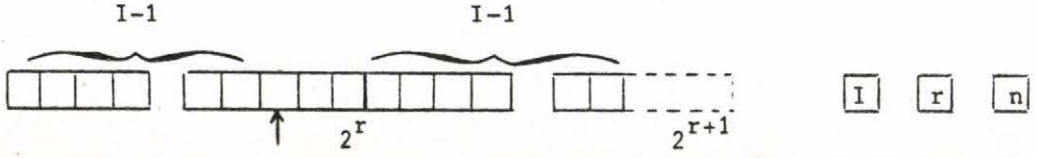


Figure 9 : Linear Splitting : Index Structure, I=1,...,2r : List L has 2^r+I-1 cells.

INSERTION

1) $n = n + 1$
2) if $n < \gamma b$ go to 6
3) otherwise ($n = \gamma b$) split bucket L_I
4) $n = 0$, $I = I + 1$
5) if $I > 2^r$, then $I = 1$, $r = r + 1$
6) end

DELETION

1) $n = n - 1$
2) if $n \geqslant 0$ go to 6
3) otherwise ($n = -1$), merge buckets L_{I-1} and L_{2^r+I-1}
4) $n = \gamma b - 1$, $I = I - 1$
5) If $I = 0$, then $r = r-1$, $I = 2^r$
6) end

It remains to define the hash function H_r which, given r and I, associates to a given key K, a unique entry in table L. From above, H_r must satisfy the following condition :

(a) $$H_{r+1}(K) = \begin{cases} \text{either } H_r(K) \\ \text{or } \quad H_r(K) + 2^r \end{cases}$$

After a bucket's chain has split, all its records, previously accessed by means of $H_r(K)$ (the buckets which contained those records were located by accessing the table entry $L[H_r(K)]$) are now accessed by computing $H_{r+1}(K)$. The latter calculation must, for each record, uniquely determine whether the $H_r(K)$th index entry or the $(H_r(K) + 2^r)$th entry has to be chosen.

Function H_r is expected to be such that when the split occurs, approximatively half of the records stay in the same bucket, and approximatively half of the records are relocated into the new bucket. The ideal function is that which, given the key distribution, satisfies the following condition :

(b) for all K, $$H_{r+1}(K) = \begin{cases} H_r(K) \text{ with probability } 1/2 \\ H_r(K) + 2^r \text{ with probability } 1/2 \end{cases}$$

It is suggested that the following hash function, employed in Virtual Hashing [4 , 5], be chosen :

(23) $$H_r(K) \equiv K \bmod 2^r + 1$$

Other functions, e.g. the multiplication function are discussed in [11]. Condition (b) will be assumed in the subsequent analysis.

We may now outline the search algorithm :

1) $L = H_0(K)$

2) $R = H_r(K)$

if $R \geq I(L)$, go to 4

3) otherwise $R = H_{r+1}(K)$

4) END (the Rth entry of the Lth table contains a bucket's address).

Before going into the performance analysis of this scheme, let us sketch the following variant : choose M = 1, suppress step 1 in the search algorithm (we have only one index list and we do not need anymore function H_0) and replace function H_r defined in Eq. (23) by the following one :

$$H_r(K) = K \bmod M \times 2^r + 1$$

3.2 - Performance analysis

The index obviously has approximatively $N/\gamma b$ cells. 3M supplementary cells are need which are much smaller in size. Observe that each of the $N/\gamma b$ cells contains only one pointer, and therefore is much smaller than the index cell of Dynamic Hashing (see Figure 1).

We now evaluate the expected secondary storage utilization. Consider subfile L (L = 1,...,M). Let α_I be the expected storage utilization when subfile L contains exactly $N_I = (1 + 2 + \ldots + 2^{r-1} + I - 1)$ records, i.e., when a split or a merge occurs, the subfile level being r, the next bucket to be split being I ($I = 1,\ldots,2^r$), (see Figure 9).

The number of primary buckets allocated to subfile L before the split (merge) is performed is equal to $2^r + I - 1$ (see Figure 9). $2(I - 1)$ buckets are accessed by employing the hash function H_{r+1}, while $(2^r - I + 1)$ buckets are accessed by means of H_r. Each of these buckets possibly has one or more overflow buckets linked to it. Denote by OV_r, the expected number of overflow buckets linked to a primary bucket accessed by means of H_r. Then α_I is given by the following expression :

$$\alpha_I = \frac{N_I/b}{2^r + I - 1 + 2(I - 1)\, OV_{r+1} + (2^r - I + 1)\, OV_r} \tag{24}$$

Assuming the distribution of the N_I records is uniform over the 2^r buckets, the probability $p_I(k)$ that bucket L_j ($j = I, I + 1, \ldots, 2^r$) and its overflow buckets contains k records is given by[6]

$$p_I(k) = \binom{N_i}{k} \left(\frac{1}{2^r}\right)^k \left(1 - \frac{1}{2^r}\right)^{N_i - k}$$

$p_I(k)$ is given by the following asymptotic approximation :

$$p_I(k) \simeq \beta_I^k \; e^{-\beta_I}/k! \tag{25}$$

where

$$\beta_I = \frac{N_I}{2^r} \simeq \gamma b \left[1 + \frac{I - 1}{2^r}\right]$$

Then

$$OV_r = \sum_{j \geq 1} j \sum_{k=jb+1}^{(j+1)b} p_I(k) \tag{26}$$

and

$$Ov_{r+1} = \sum_{j \geq 1} j \sum_{k=jb+1}^{(j+1)b} q_I(k) \tag{27}$$

where

$$q_I(k) = \binom{N_i}{k} \left(\frac{1}{2^{r+1}}\right)^k \left(1 - \frac{1}{2^{r+1}}\right)^{N_i - k} \simeq \left(\frac{\beta_I}{2}\right)^k \; e^{-\beta_I/2}/k! \tag{28}$$

---

6) An analysis of Hash-Coding with Separate Chaining due to Van der Pool [12] is reported in [1, p. 537]. It is implied in [1] that all buckets share a common overflow area, while in the present paper, each bucket owns a separate overflow area.

Combining Eqs. (24) to (28) reduces to the following expression :

(29)

$$\alpha_I = \frac{1}{1 + \frac{1}{2^r+I-1}\left[2(I-1)\sum_{j>0} j \sum_{k=jb+1}^{(j+1)b} \frac{(\beta_I/2)^k e^{-\beta_I/2}}{k!} + (2^r-I+1)\sum_{j>0} j \sum_{k=jb+1}^{(j+1)b} \frac{(\beta_I)^k e^{-\beta_I}}{k!}\right]^\gamma}$$

As the file steadily grows α_I will eventually converge to a value independent of I. Steady state is rather rapidly reached as shown in Table 3 : the difference between the minimum value and the maximum value of α_r is less than 2 %, if the subfile contains more than 10 000 records (if $\gamma = 1$, $b = r = 10$, then $N_I > 10\ 240$). If $\gamma = 1$, α_I is, on the average, only slightly greater than Ln2, but if $\gamma \geq 2$, storage utilization is high ($> .81$).

γ \ I	1	200	400	600	800	1024
1	.705	.686	.698	.713	.715	.705
2	.818	.814	.812	.814	.818	.819
3	.8698	.8701	.8701	.8690	.8694	.8699

Table 3 : Linear Splitting : Expected Storage Utilization as a function of I, b = 10, r = 10.

As far as the number of accesses to secondary storage is concerned, this scheme is rather unfair, since some records (stored in L_I, where I is small) are accessed very fast, while other records (I large) may require a large number of accesses. Indeed, just after the split of bucket L_1 (I = 1), on the average $\beta_1/2 = \frac{\gamma b}{2}$ records are stored in it, while just after the split of bucket L_{2^r} ($I = 2^r$) on the average $\beta_{2^r}/2 = \gamma b$ records are stored in it (and $\beta_{2^r} = 2\gamma b$ records before it is split !).

If we take the average over the 2^r buckets, we get the following performance measure (when the file contains N_I records) :

$$T_I = \frac{1}{N_I}\left[2(I-1)\ T_{r+1} + (2^r-I+1)\ T_r\right] \tag{30}$$

where T_r is the expected number of accesses required to find all the records stored in a buckets' chain, given this chain is accessed by means of H_r. T_r is given by the following expression :

$$T_r = \beta_I + \sum_{k=b+1}^{2b}(k-b)p_I(k) + \sum_{k=2b+1}^{3b}[b+2(k-2b)]p_I(k) + \sum_{k=3b+1}^{4b}[b+2b+3(k-3b)]p_I(k)+\ldots$$

$$T_r = \beta_I + \sum_{j\geq 1} j \sum_{k=jb+1}^{(j+1)b}\left[k - \frac{(j+1)b}{2}\right] p_I(k) \tag{31}$$

Similarly,

$$T_{r+1} = \frac{\beta_I}{2} + \sum_{j\geq 1} \sum_{k=jb+1}^{(j+1)b}\left[k - \frac{(j+1)b}{2}\right] q_I(k) \tag{32}$$

Combining Eqs. (30), (31) and (32) reduces to the following expression :

$$T_I = 1 + \frac{1}{\beta_I}\left\{2(I-1) \sum_{j\geq 1} j \sum_{k=jb+1}^{(j+1)b}\left[k - \frac{(j+1)b}{2}\right]p_I(k) + (2^r-I+1) \sum_{j\geq 1} j \sum_{k=jb+1}^{(j+1)b}\left[k - \frac{(j+1)b}{2}\right]q_I(k)\right\} \tag{33}$$

where $p_I(k)$ and $q_I(k)$ are respectively given by Eqs. (25) and (28).

T_I oscillates more significantly (as a function of I) than α_I does (Table 4). Note that the extrema of T_I do not correspond to those of α_I (see Table 3). From Table 4, it is clear that the access performance degradation is not important since for a storage utilization approximatively equal to .81 (i.e. $\gamma = 2$, see Table 3). T_I 1.71.

γ \ I	1	200	400	600	800	1024
1	1.125	1.174	1.197	1.194	1.171	1.125
2	1.591	1.669	1.711	1.714	1.677	1.595
3	2.077	2.195	2.256	2.259	2.206	2.083

Table 4 : Linear Splitting :Expected number of accesses to secondary storage as a function of I, b = 10, r = 10.

Figure 10 illustrates more precisely the trade-off between fast retrieval and high utilization. The curve is labelled by the corresponding values of γ. The performance of Dynamic Hashing (Figure 4) is plotted on the same figure. Clearly, Linear Splitting performs almost as well as Dynamic Hashing with Deferred Splitting.

Moreover, the number of index nodes is asymptotically smaller in the former case than in the latter one, for a given (T, α) performance : e.g. if $\gamma = 2$, $\beta = 3$, then the index size of the scheme studied in Section 2.2 is, from Eq. (2) $2N/\beta bLn2 - M = N/1.04b - M$, while in the present scheme, the index size is equal to $N/2b + 3M$; Besides, the index node size of Dynamic Hashing with Deferred Splitting is much larger (Figure 1) than the index node size of the present scheme which contains only one pointer. Thus Linear Splitting requires much less storage for the index than Dynamic Hashing with Deferred Splitting does.

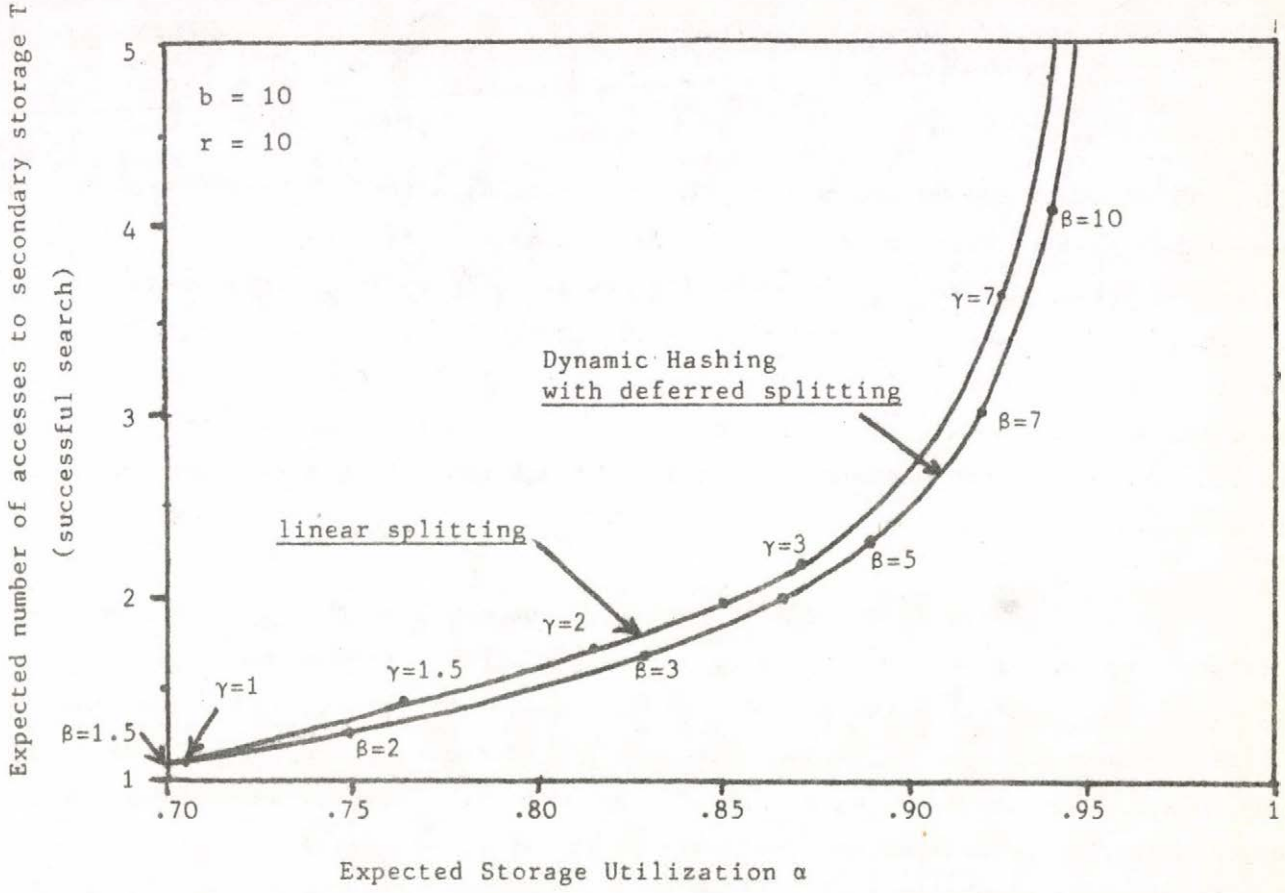


Figure 10 : T vs α : Comparison between Dynamic Hashing with Deferred Splitting and Linear Splitting

4 - CONCLUSION

In this paper, we have proposed two new access techniques based on hash-coding, for files whose size may vary by large factors. The goal pursued was to increase storage utilization which is approximatively equal to Ln2 only, in the file organizations previously proposed [2, 3].

The first scheme studied, a variant of Larson's Dynamic Hashing scheme [2] in which splitting of any bucket is deferred until a certain number of overflow records have been inserted, definitively improves the file performance.

Indeed if we allow any bucket to overflow twice its capacity, the index size is three times smaller than in Larson's scheme, an .83 expected storage utilization is reached, while the expected number of accesses to secondary storage required to find a record (successful search) is equal to 1.64, provided the index is available in main storage (Figure 4 : $\beta = 3$) and in the worst case, at most three accesses are needed. Another variant is suggested (Section 2.3) which provides an even better performance. However this results in a more complex storage management.

If the file steadily grows on, the index of the first scheme will eventually be partly stored in secondary storage. Because of the index tree structure, retrieval of a record may then be considerably slowed down. The second scheme, nicknamed Linear Splitting, has been introduced in order to avoid such a degradation. It employs a much smaller index and thus the file supports a larger number of records before the index overflows on secondary storage. Besides, even though part of the index is stored in secondary storage, searching for an index entry is fast, because of the linear structure of the index. Provided the index is available in main storage, Dynamic Hashing with linear splitting performs quite as well as the first scheme (Figure 10).

Finally, it is worth noting that both schemes keep the searching and updating algorithms simple.

ACKNOWLEDGMENT

The author wishes to thank W. Litwin. Helpful discussions with him led to the definition of the second scheme.

REFERENCES

[1] D.E. Knuth - The art of computer programming Vol. 3 - Sorting and Searching - Addison-Wesley - Reading - Mass. - 1973.

[2] P.A. Larson - "Dynamic Hashing", BIT, vol. 18, 1978, pp. 184-201.

[3] R. Fagin, J. Nievergelt, N. Pippenger, H.R. Strong - "Extendible Hashing - A Fast Access Method For Dynamic Files", IBM Research Report RJ2305, San Jose, California, July 31, 1978.

[4] W. Litwin - "Auto-Structuration d'un fichier : méthodologie, organisation d'accès, extension du hash-coding", Institut de Programmation res. rep. 77/11, Paris, April 1977.

[5] W. Litwin - "Virtual Hashing : a dynamically changing hashing", Proceedings of the 4th Conference on very large data bases, West-Berlin, September 13-15, 1978, pp. 517-523.

[6] D.G. Keehn, J.O. Lacy - "VSAM data set design parameters", IBM Syst. J., N° 3, 1974, pp. 186-212.

[7] T. Nakamura, T. Mizoguchi - "An analysis of storage utilization factor in block split data structuring scheme", Proceedings of the 4th Conference on very large data bases, West-Berlin, September 13-15, 1978, pp. 489-495.

[8] A.C. Yao - "On random 2-3 Trees", Acta Informatica, N° 9, 1978, pp. 159-170.

[9] W. Litwin - "Linear Virtual Hashing : a new tool for files and tables implementation", invited IFIP TC-2 Working conference on data base architecture, Venice, June 26-29, 1979.

[10] M. Stonebracker, E. Wong, P. Kreps, G. Held - "The Design and Implementation of INGRES", ACM Transactions on Database Systems, vol. 1, N° 3, Septembre 1976, pp. 189-222.

[11] D.E. Knuth - The art of computer programming, vol. 2 : Semi-numerical algorithms, Addison-Wesley, Reading, Mass, 1969.

[12] J.A. Van der Pool - "Optimum Storage Allocation for Initial Londing of a File", IBM J. Res. Develop., November 1972, pp. 579-586.

BEHAVIOUR MODELS OF COMPUTER SYSTEMS

G.Bergholz
GDR

0. Preliminary remarks

In general, this paper is oriented on the development of models for optional computer systems, the crucial point being the analysis of operating systems. There, the examples used in the representation originate from plant computer applications. This way, however, the applicability of the methods used does not remain confined to the field of plant computers. The methods are also applicable to batch processing systems, time-sharing systems and computer networks. This contribution will provide an orientation for the development and application of behaviour models for computer systems, and the general representations will be supported by illustrations of simple examples.
Investigations conducted by the author serve as the basis for the statements. A major part of the findings was already published in the papers /1/, /2/, /3/, /4/, /5/, /6/, /7/, and elsewhere.

1. Efficiency evaluation of computer systems

Performance and elapsed time of the jobs are used as the most important values for estimating the efficiency of a computer system. There, the maximum job throughput of the computer system is comprehended as performance. Mostly, the elapsed time of the jobs is the time interval elapsing

between the activation and the termination of a computer job. In general, the mean response time and sometimes also the distribution of the elapsed time of the job are examined there. In many cases, the elapsed time corresponds to the response time of the plant computer system.

Independent influencing factors on which the values for estimating the efficiency are dependent are:

- the structure of the computer system, in this paper described by the transaction flow scheme /TFS/, see Section 2.
- the characteristics of the arriving flow of the job flow expressed in general by the distribution, $A(t)$, and in particular by the intensity, λ , of the arriving flow of jobs.
- the characteristics of service time, in general described by the distribution, $B(t)$, and in particular by the mean service time, $E(t_b)$.

In the transaction flow scheme, besides that, branching probabilities, $P_i\ (i=1,2,\ldots,n)$, at n branching points within the TFS are playing a part as independent influencing factors.

The behaviour model of a computer system describes the relation between the independent influencing factors and the values for estimating the efficiency of the computer system. It serves as a means of efficiency evaluation of the computer system. There, the overall objective of the investigation always consists in the improvement of the efficiency of the computer system. The measures necessary for this may cover the following fields:

- the improvement of the hardware contingencies of a computer system in designing new computers
- the improvement of the contingencies of the operating system in developing new operating systems
- the improvement of the hardware configuration of a certain computer installation
- the regeneration of the operating system for a certain computer installation
- the improvement of the application programs with respect to the efficiency of the computer system.

There, especially the first four fields are well to the fore, because practically always they can be realized by specialists of computer technique. As an application programmer, however, I would like to concentrate upon the problems of application and to consider as few specific problems of computer technique as possible.

2. Description of the operating mode of a computer system

2.1. System elements and their states

A computer system serves jobs. In plant computers, we talk about real-time jobs. /1/, /3/. Thus, we regard the job as an external work unit /at the interface computer - user/. The internal work unit of the computer system is regarded as a transaction. Due to the possible parallelism of the job service, the latter is necessary. It is assumed that an individual transaction will always be served sequentially, i.e. the operations necessary for it to be served will occur one after another as to time, and not parallel to another. Because a job is mostly served parallel or at least quasiparallel, and because the operations necessary for it to be served have to be carried out parallel, it will often consists of several transactions. The ensemble of transactions involved in a job is called a transaction

unit. Serving the jobs, resources are necessary. Resources such as these are the central processing unit /CP dynamical main memory regions; resources of information exchange; resident, not reentrant programs; data sets; and another resources.

Basing on these concrete conditions, we realize a first abstraction and assume the transactions and the resources as element classes. In these, the transactions are active elements and the resources are inactive ones. The operating mode of the computer system is described by the interaction of transactions and resources. This interaction is realized by changes of state of the transactions and the resources. We assume that the compute system consists of a constant number, q, of resources, Ri(i=1,2,...q), and a variable number of transactions, Fj(j=1,2,...).

Let us consider now the most important states of the transactions and resources, the change of which describes the interaction of the system elements. The existent state of a transaction, Fj(j=1,2,...), has the following state values:

- 0 - not existent
- pl - planned /timeless intermediate state/
- wa - waiting /in a queue in fromt of a resource/
- be - serving /by resources/
- sy - synchronizing /i.e., waiting for an event, also called suspended /

In Fig.1., the state graph for the existent state of a transaction, Fj(j=1,2,...), is given.

The busy state, R_i, of a resource, Ri(i=1,2,...q), has the following coordination of values

$$R_i = \begin{cases} 0 & R_i \text{ idle} \\ 1 & R_i \text{ seized once} \\ \vdots & \\ K_i & R_i \quad K_i \text{ times seized} \end{cases} \qquad (i=1,2,...q) \qquad /1/$$

There, K_i is the channel number of the resource, Ri(i=1,2,...9).
The change of state is put up with

$$R_i := R_i + \Delta R_i \qquad /2/$$

where

$$\Delta R_i = \begin{cases} 1 & \text{seizure of Ri} \\ -1 & \text{release of Ri} \end{cases} \qquad /3/$$

The state graph for the busy state of the resource, Ri(i=1,2,...9) is presented in Fig.2.

2.2. Transaction flow scheme

Now, we introduce operation blocks for the different changes of state. There, the changes of state for the entry into and the exit out of the time-subjected existent states of a transaction, Fj (j=1,2,..), SERVE, WAIT and SYNCHRONIZE are combined to one operation block, in each case. Names, symbols and the respective changes of state of the introduced operation blocks are given in Tab.1.

The first two operation blocks serve the generation of transactions. There, the block GENERATE corresponds to the generation of a sequence of transactions in definite time intervals, T_a. The block SPLIT describes the generation of a second transaction at the moment when a transaction passes through this block. The next two blocks serve the termination of the transactions arriving there. In the block TERMINATE, every transaction entering the block is terminated. With the block ASSEMBLE, it is assumed that in each case there is always a block SYNCHRONIZE in front of it at both entries. The transaction that arrives first in one of these two blocks

SYNCHRONIZE will remain in this block until a transaction arrives in the second block SYNCHRONIZE. If there is at least one transaction each in the two blocks SYNCHRONIZE, one transaction each from the two blocks SYNCHRONIZE are transferred to the block ASSEMBLE, and one of them is terminated. In their effect, the two blocks BRANCH and COMBINE check with the respective blocks in the program flowcharts. In this, COMBINE is not regarded as a special block in program flowcharts. The three block SERVE, WAIT and SYNCHRONIZE are time-subjected: i.e., a transaction generally stays in the respective state for a certain time. There, the time, T_b, of the block SERVE /service time/ is presumed as being given, while the time, T_w, of the block WAIT /waiting time/ and the time, T_s, of the block SYNCHRONIZE /synchronizing time/ depend on the occurrence of certain events. While all operation blocks considered so far refer to the change of the transaction state, the two blocks SEIZE and RELEASE are directed to the change of the busy state of the resources entered into these blocks. In this, it has to be observed that there must be a block WAIT or /in case of transaction loss/ a block BRANCH in front of a block SEIZE. If one of the resources characterized by the block SEIZE is completely seized - i.e., no channel is idle any more - then, a transaction contained in the block WAIT or in the block BRANCH cannot enter the block SEIZE, and it is either stopped by the block WAIT, or it is conducted past the respective block SEIZE by the block BRANCH.

Altogether, the following approach is true for the transaction flow scheme /TFS/. The individual trans-

actions pass through the operation blocks corresponding to the arrows, one after another and depending on the states. When a transaction passes through an operation block, the change of state that corresponds to this block is realized.

In accordance with Dijkstra, the flow of a transaction with the pertinent changes of state may be called a sequential process.

Different sequential processes may run parallel to each other. There, we distinguish between the internal and the external parallelism. Internal parallelism means parallelism between the sequential processes /transaction flows/ within a job. Parallelism between the sequential processes of different jobs is referred to as external parallelism.
In Fig.3., the example of a transaction flow scheme for the interaction of an application program with an information exchange resource is represented. By means of this TFS, the three resources, R1, R2 and R3 are comprehended. There, we assume R1 to be the processor, R2 to be the application program, and R3 to be the information exchange resource. The respective times of waiting, serving and synchronizing are identified in the representation.
It is assumed that after the service time, T_{b1}, the information exchange is requested by the application program R2, and the results of the information exchange in the service time, T_{b2}, of the application program are required. During the information exchange, the processor may be utilized for other programs, what is expressed by RELEASE of R1 after the initiation of an information exchange. In order to realize the section with the service time after completed information ex-

change, a synchronization of the two sequential processes is realized via the synchronizing blocks with the synchronizing times, T_{s1} and T_{s2}, and the assembling blocks, and the transaction of the transfer process is terminated.

2.3. Making the transaction flow scheme

For the better understanding of the time course, it will be appropriate to mark the time-subjected blocks of the type WAIT, SERVE and SYNCHRONIZE. In this, marking the time-subjected block consists in specifying the number of transactions existing in this block at the moment of observation. If we provide the time-subjected blocks with names, the marking, M(i), of the system may also be given on this basis. There, we imagine the time course to be pulsed where each pulse is related to a section of unchanged state. Timeless pulses are permitted. There, the variable, i, is the pulse number.

Basing on this, a marking table for the computer system may be prepared in which the individual lines represent the marking, M(i), of the system in the individual pulses; and the columns represent the temporal marking sequences of the respective time-subjected blocks. In Tab.2, an example of the marking table for the transaction flow scheme in Fig.3 is given. The indices of the times of waiting, serving and synchronizing are used as names of the time-subjected blocks. It has to be observed that, depending on the time intervals of arrival of the generation of transactions and on the waiting times for the transaction flow scheme in Fig.3, there may be a multitude of possible marking tables. The marking tables given in Tab.2. only represent one realization of the random process that is structurally described by the transaction flow scheme. By means of the pulses 1 and 3, transactions

are generated in the operation block GENERATE, and by means of pulse 4, they are generated in the operation block SPLIT, in this realization. A termination of the transactions is realized by the pulses 10 and 17 via TERMINATE, and by the pulses 8 and 15 via ASSEMBLE.

2.4. Transition from the transaction flow scheme to a Petri-net

Recently, Petri-nets have been used for certain, mostly deterministic investigations of computer systems. /8/ Petri-nets permit the investigation of system deadlocks in computer systems. Though in the foreground of this paper there is the random analysis with the derivation of models for estimating the efficiency of a computer system, it will be useful, however, to establish the connections between the transaction flow scheme and the Petri-net. In particular, the connection will be important if both the evaluation regarding efficiency and the evaluation regarding system deadlocks are to be pursued.
In Tab.3., the coordination of Petri-net sections and TFS sections is represented. It has to be observed that inusing this table, an oriented graph is formed as an intermediate stage that may contain adjacent event nodes, something that is not allowed in Petri-nets. By combining adjacent event nodes, the Petri-net that corresponds to the transaction flow scheme will be obtained.

3. Random behaviour models

The transaction flow scheme forms the structural basis for the specification of behaviour models. In a behaviour model, we observe the time behaviour of the computer system. Therefore, especially the waiting times,

T_w, the service times, T_b, and the synchronizing times, T_s, are to be considered. We assume these times to be random variables, and we apply the probability theory and the mathematical statistics to the transaction flow scheme.

Equally and complementing one another, the following methods of behaviour analysis have been generally accepted:

- the statistical evaluation on the basis of system measurements,
- the statistical evaluation by means of event-oriented simulation, and
- the probabilistic evaluation, utilizing service models and random nets.

Under the management of the author, investigations have been conducted in all three directions, the latter two methods being in the foreground see /1/, /3/, /4/, /5/, /6/, and /7/.

In these, evaluations of performance as well as evaluations of response time have been carried out. Here, we confine ourselves to the evaluation of performance, utilizing probabilistic methods.

4. Performance evaluation of computer systems

The performance evaluation of a computer system is of great and direct importance for batch processing systems. For plant computers, the elapsed time is mostly of greater importance. But also in this case when determining the response time or the elapsed time is the main concern, performance evaluation is mostly carried out as as inter-

mediate step of investigation. The performance evaluation is connected with the determination of the resources emerging as bottlenecks of the computer system. Thus, the performance evaluation may be utilized as a means of model simplification, by only considering those resources service-theoretically for the evaluation of the elapsed time that are the bottlenecks of the system. This mostly results in a considerable simplification of the evaluation of the elapsed time without substantially influencing accuracy.

In the performance evaluation, we consider all resources Ri(i= 1,2,...,q), and for each resource we introduce the rate of utilization

$$\eta_i = \lambda_i \, E(T_{1i}) \quad (i= 1,2,...,q) \qquad /4/$$

There, λ_i is the intensity of the arriving flow of transactions at the entry of the resource, Ri, and E(Tei) is the mean busy time of the resource, Ri, by a transaction. Moreover, we introduce the performance G_i, of the resource Ri as

$$G_i = 1/E(T_{1i}) \qquad (i= 1,2,..,q) \qquad /5/$$

The performance G_s of the computer system is calculated according to

$$G = \text{Min} \{ G_i/\varepsilon_{ij} \quad (i= 1,2,...,q)\} \qquad /6/$$

There ε_i is the coefficient of the intensity transfer between the entry of the system and the entry of the resource Ri and Gi is the performance of the resource Ri.

Statement /6/ shows that G is always determined by one or more equal terms G_i/ε_1. The resources R1 for which

$$G = G_1/\varepsilon_1$$

is true, i.e. the resources determining the performance of the system are referred to as bottlenecks.

Analyzing the system described by the TFS in Fig.3, the performance

$$G = \frac{1}{E(T_{b1}) + E(T_{b2}) + E(T_{b3}) + E(T_{w3}) + E(T_{w2})} \quad /8/$$

will result. There the resource R2, i.e. the application program, is the bottleneck. This finding may be generalized in this sense that the CPU may be poorly utilized but nevertheless there is the possibility of overloading a program caused by transfer, waiting and synchronizing times.

5. Applying the performance evaluation for determining the bottlenecks in a plant computer installation

Utilizing the method of performance evaluation described in Section 4, the performance evaluation for a plant computer installation "tandem roll train" has been carried out on the basis of the data obtained by means of the applications engineering.

In Tab.4, the findings thus obtained are represented. For the resources considered, the utilization of the respective resource with an intensity of the arriving flow that

corresponds to the conditions in practice is given there, and the performance of the system on condition that the resource considered were bottleneck is given, too.

It reveals that the two resources AS 1/2 /an information exchange resource for the logging print/ and TEBE /an application program for the preparation of the logging print/ are the bottlenecks. Besides that, the second information exchange resource AS 1/1 closely approaches its utilization if AS 1/2 is utilized. For this case of application, the employed resources of the plant computer will meet the requirements. But, in order to increase the accuracy and to adapt to the possibly to be expected higher demands regarding the rate of arrival of the real--time jobs, measures should be taken to relieve the three resources mentioned above. In /2/, adequate recommendations are made on the basis of the evaluation of the elapsed time.

References

/1/ Bergholz, G: Zur Analyse des Multiprogrammbetriebs in einer Prozessrechenanlage. Rechentechnik/Datenverarbeitung 11 /1975/, Beiheft 3

/2/ Richter, F. und Bergholz, G.: Zur Effektivitätsanalyse von Prozessrechnern. Konferenzmaterialen zum Symposium Informationsverarbeitung anlässlich des zehnjährigen Bestehens der Sektion Informationsverarbeitung /1979/

/3/ Bergholz, G.: Zur Ermittlung der Forderungsstromintensitäten in einem Echtzeitoperationssystem für Prozessrechenanlagen. Wiss.Z.Techn.Univers. Dresden 24 /1975/, Heft 3/4

/4/ Bergholz, G.: Verhaltensmodelle von Prozessrechnern Akademie-Verlag Berlin /in preparation/.

/5/ Bergholz, G.: Zur Bestimmung der Reaktionszeit von Prozessrechenanlagen. Közlemények 18/77 AdW der UVR Budapest

/6/ Bergholz, G.: Zur Analyse des Echtzeitbetriebs von Prozessrechnern ZfR-Informationen AdW der DDR, ZfR - 01.77

/7/ Bergholz,G.: Ähnlichkeiten zwischen Verwilzeit-modellen eines Prozessrechners und Signalübertragungssystemen, msr 21 /1978/ H.1o.

/8/ Schumacher, F.:Modelling and Simulation of Computer Systems with Simulation Nets Selected Papers on Operating Systems Edited by M.Arató, Budapest 1978.

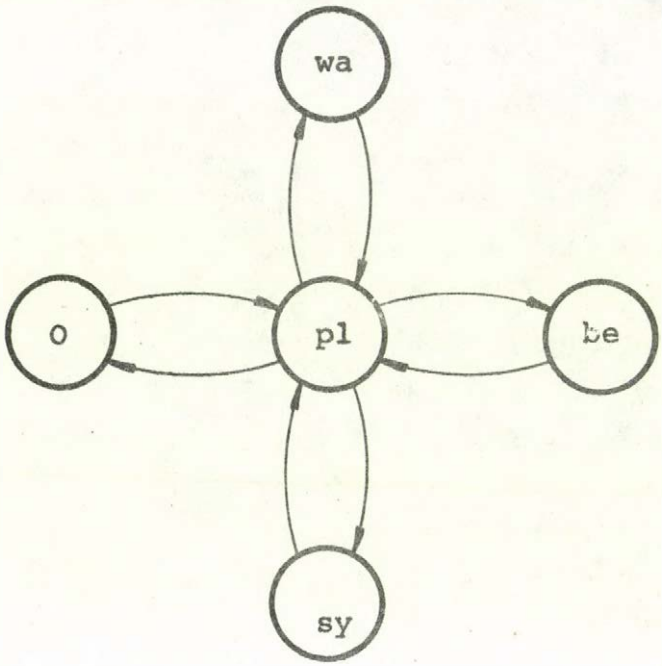


Figure 1

STATE GRAPH FOR THE EXISTENT STATE OF A TRANSACTION

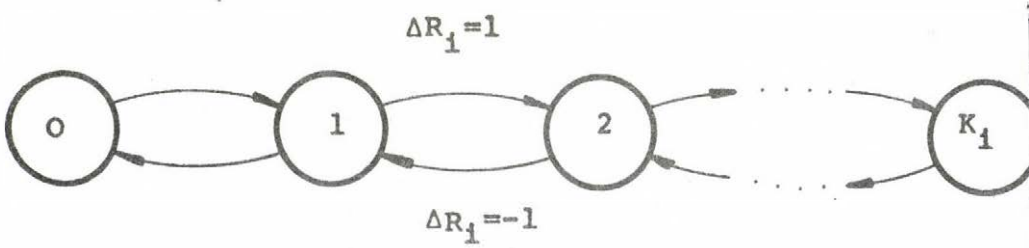


Figure 2

STATE GRAPH FOR THE BUSY STATE OF A RESOURCE

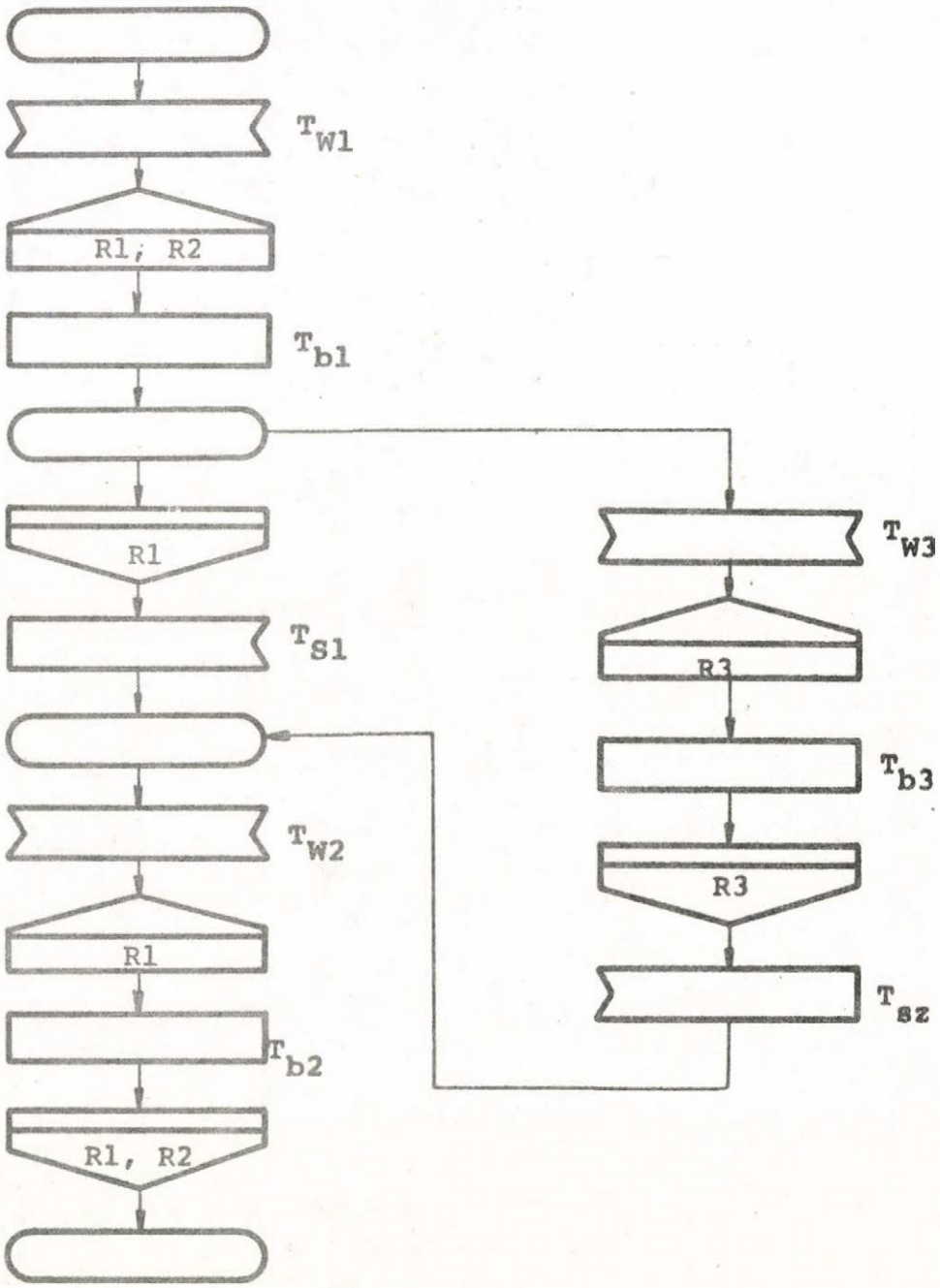


Figure 3

EXAMPLE FOR A TRANSACTION FLOW SCHEME FOR THE INTERACTION OF A PROGRAM AND AN INFORMATION EXCHANGE RESOURCE

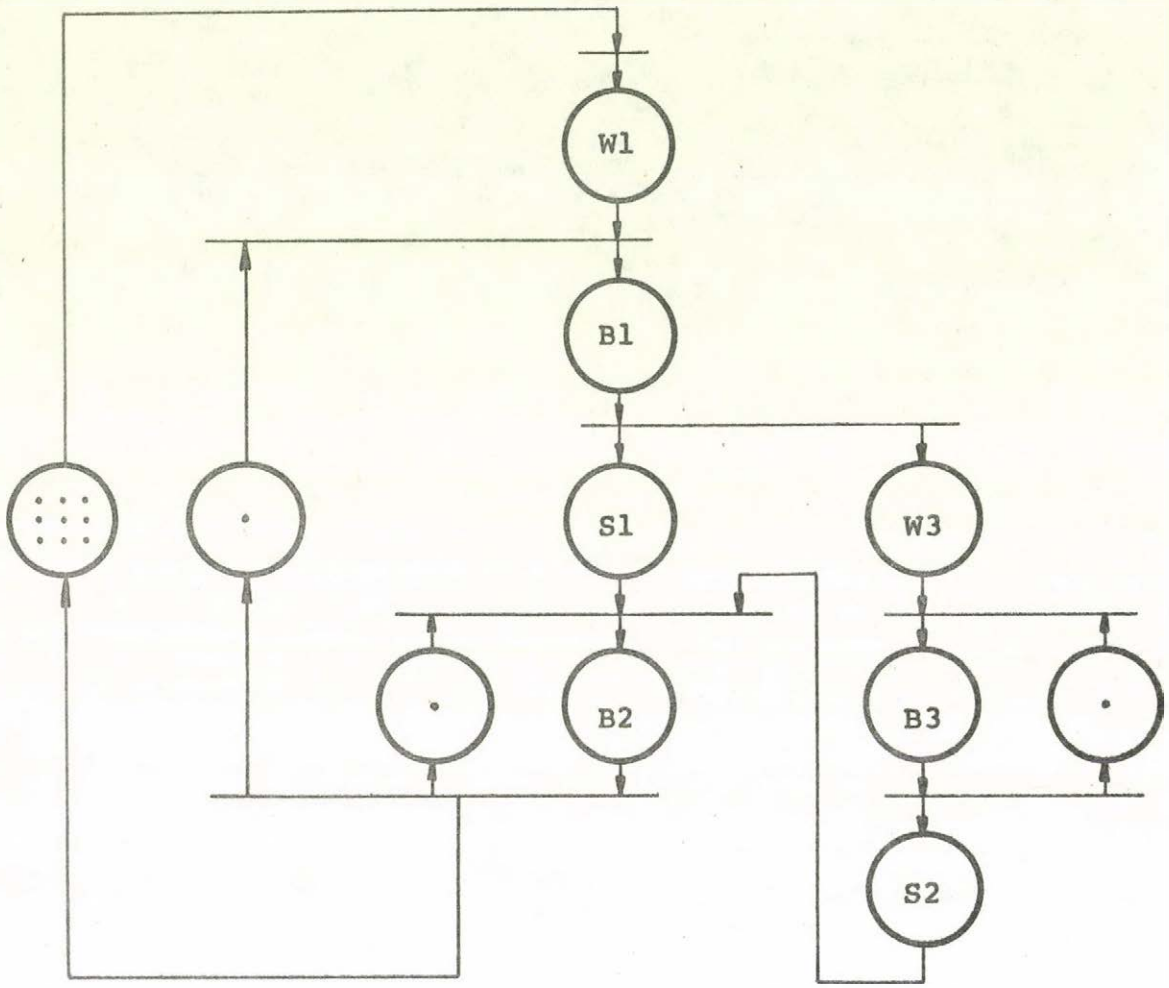


Figure 4

PETRI-NET FOR THE SYSTEM DESCRIBED BY THE TRANSACTION FLOW SYSTEM IN FIGURE 3

TABLE 1: OPERATION BLOCKS FOR STATES AND CHANGES OF STATE

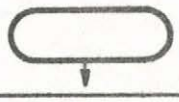
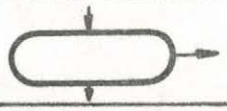
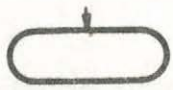
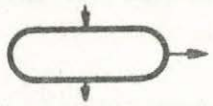
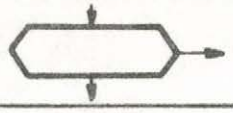
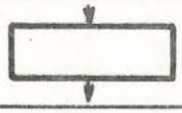
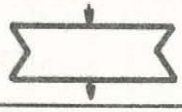
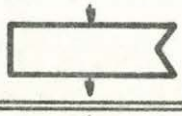
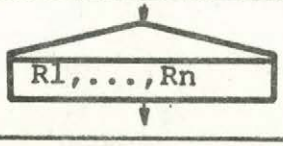
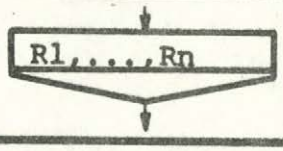
NAME	SYMBOL	CHANGE OF STATE
GENERATE		0 pl
SPLIT		/for the generated transactions/
TERMINATE		pl 0
ASSEMBLE		/for the terminated transaction/
BRANCH		only change of the running state of the transaction
COMBINE		"
SERVE		enter: pl be exit: be pl
WAIT		pl wa wa pl
SYNCHRONIZE		pl sy sy pl
SEIZE		$R_i := R_i + 1$ /i=1,2,...,n/
RELEASE		$R_i := R_i - 1$ /i=1,2,...,n/

Table 2: A marking table for the transaction flow scheme in Fig.3. when serving 2 jobs

pulse number	time-subjected blocks							
	W1	B1	S1	W2	B2	W3	B3	S2
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	1	0	1	0	0	1	0	0
5	1	0	1	0	0	1	0	0
6	1	0	1	0	0	0	1	0
7	1	0	1	0	0	0	0	1
8	1	0	0	1	0	0	0	0
9	1	0	0	0	1	0	0	0
10	1	0	0	0	0	0	0	0
11	0	1	0	0	0	0	0	0
12	0	0	1	0	0	1	0	0
13	0	0	1	0	0	0	1	0
14	0	0	1	0	0	0	0	1
15	0	0	0	1	0	0	0	0
16	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0

TABLE 3: TRANSFER OF A TRANSACTION FLOW SCHEME IN A PETRI-NET

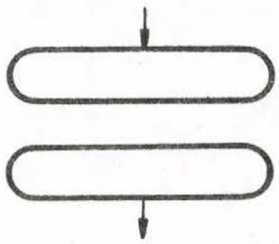
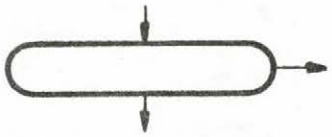
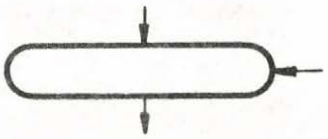
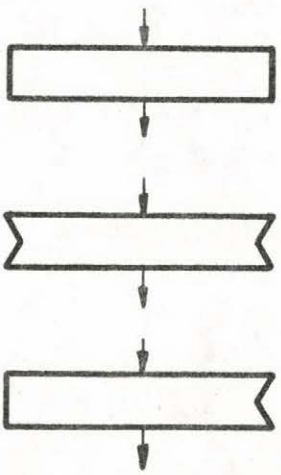
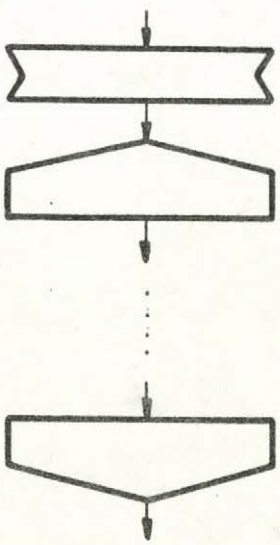
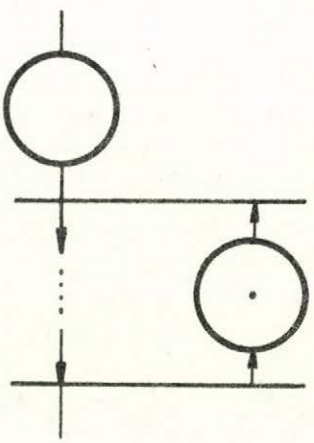
Contents	TFS	PETRI-NET
TERMINATE GENERATE		
SPLIT		
ASSEMBLE		
TIME BLOCKS		
SEIZURE OF RESOURCES		

Table 4: Evaluation of the bottleneck of a plant computer installation in a tandem roll train

resource	utilization (= 1,0 s^{-1})	performance s^{-1}
MASK	0,0006	1670
ZERF	0,0006	1670
ZANE	0,0070	143
ZYNE	0,o216	46,3
MEAU	0,0022	463
SPAU	0,0078	128
STAU	0,0043	232
STIB	0,1150	8,7
ERBU	0,0640	15,6
BUBE	0,0351	28,5
TEBE	0,4210	2,37
AER	0,0415	24,1
AS1/1	0,3686	2,71
AS1/2	0,4250	2,35

A PROGRAM PACKAGE ARCHITECTURE FOR COMPUTER NETWORK MEASUREMENT

D.AMBRÓZY, A.SZABÓ, K.TARNAY

Central Research Institute for Physics
Budapest, Hungary

ABSTRACT

This paper deals with general principles of computer network measurement, with the scheme and architecture of a monitor and with some already implemented modules:

1. The RTS/BSC module provides an interface to a BSC oriented telecommunication line/network
2. The BSCMON module is responsible for basic measuring capabilities
3. The simulator executes simulation experiments.

A PROGRAM PACKAGE ARCHITECTURE FOR COMPUTER NETWORK MEASUREMENT

D.AMBRÓZY, A.SZABÓ, K.TARNAY

Central Research Institute for Physics
Budapest, Hungary

1. INTRODUCTION TO COMPUTER NETWORK MEASUREMENTS

1.1. Requirements

Nowadays computer networks are spreading all over the world and therefore their measurement came into prominence. In the Central Research Institute for Physics in Hungary a program package for computer network measurement is under development. We want to speak on its architecture and about some modules.

Which kind of requirements can be raised in connection with the monitor?

- a hierarchical architecture
- no interference with the operation of the computer network
- measurement of optional network parameters
- well separable modules
- expandable in a simple way
- easily applicable to different computer networks

1.2. The hierarchy of measurements

The hierarchy of our program package contains three levels:

- diagnostics
- performance measurement
- analytic measurement

The purpose of diagnostic measurements is the control of correct operation of computer networks, the detection and diagnostics of errors. The operation of computer network is characterized by events of different types. The main events can be classified into two groups:

- current network configuration
- current operational status

The performance of a computer network is determined by the quality and quantity of services for a given workload. Some of the parameters - to be measured - are for instance:

- time /time necessary to set up and disconnect a connection, average delay time by several routing policies, time duration for using the resources etc./
- throughput /number of transmitted and processed messages, parameter: packets per message, number of transmitted bits etc./
- capacity of the node stores /main and background stores/
- queues /number of messages, packets, bits in different waiting queues/

The highest hierarchical level is the analytical measurement: the analysis of normal operation under dynamically changing conditions. The analysis is an effective tool for understanding processes in computer communication, their impact on the system performance and for proving relationships between network parameters.

2. THE DRAFT OF THE PROGRAM PACKAGE

2.1. The scheme of the program package

The task of the monitor program package is the measurement of deterministic and stochastic parameters of computer network. The monitor (Fig.1) as a host is connected to a single node of the network. The connection is implemented through a communications subsystem which forwards the signals from the control unit and, optionally, the artificial workload to the network. The signals - to be measured - get to a sampler, the frequency and the time- or event-controlled start of sampling are regulated by the control unit. The sampled signals are the inputs of the measuring subsystem. The components of the measuring subsystem depend on the measuring methods and on the parameters to be measured. After the required computing the evaluation routine keeps a diary on the important results, gives an alarm if it is necessary and collects information for further processing. The workload generator generates different artificial workloads for the network. This workload is optional, we can measure the network also under real normal operational conditions. Fig.2. shows the scheme of workload generator, it contains four separable modules:

- computer network simulator
- message mix
- operational conditions
- protocol generator

2.2. The components of the measuring subsystem

The parameters of the computer network are deterministic or stochastic. The first decision of the measuring subsystem, is to be seen in Fig.3.

After a second and a third decision, if the parameter is deterministic we can perform the measurements as follows:

- an event detector routine
- a counter routine
- a time measurement routine

If the parameter is stochastic,two decisions are possible.
It has to be decided whether or not it is

- a density function

if not, whether or not it is

- an autocorrelation function?

We have three possibilities depending on the answer to the previous questions:

- a histogram routine in the first case
- an autocorrelation routine in the second case
- a crosscorrelation routine in the third case

The output from the measurement subsystem is the input to the evaluation unit.

2.3. Sample modules and their relationships

Figure 4. shows the architecture of some implemented modules. On telecommunication level 1 and 2 two modules are implemented: the basic telecommunicationssubsystem RTS/BSC (see Chapter 3) which is the measured and the basic measurement module BSCMON (see Chapter 4) which is the measuring part, on level 3 and 4 the network simulator is already implemented and can use the RTS/BSC on line (see Chapter 6). The evaluation of the measurement results is described in Chapter 5.

3. RTS/BSC - THE TELECOMMUNICATIONS SUBSYSTEM

The RTS/BSC package is an implementation of IBM's standard Binary Synchronous Communications (BSC) telecommunication protocol into the OS/8, RTS/8 and COS 300 operating system of Digital Equipment's PDP-8 small computer.

3.1. Telecommunication protocol level 1 - physical line handling

The package implements a synchronous procedure on physical line handling level with bit and character synchronisation which is re-established for each transmission. The package uses half-duplex or full-duplex line/modem over privately owned, leased or switched lines. The package drives a synchronous interface and modem which conform to CCITT V. 24 recommendations. Line speed from 600 to 9600 baud allowed.

3.2. Telecommunication protocol level 2 - line protocol

RTS/BSC implements the point-to-point or multipoint operation procedure. In multipoint environment the small computer can be used on a multidrop line as a control or as a tributary station of a star network. The control station can handle up to 32, the tributary station up to 8 logical addresses. Multiple stations can be simulated with multiple modems, interfaces and copies of the RTS/BSC on a single computer with stations working independently or connected together to simulate point-to-point connection. The algorithm can handle ASCII, EBCDIC or transparent (any) data format. The implemented algorithm enables the computer to be connected to:

- IBM or SIEMENS mainframe using BSC (or MSV1, MSV2) telecommunication procedure;
- to an IBM 2780, IBM 3780, IBM 2770, IBM 3270 telecommunication terminal or a SIEMENS Transdata 840X terminal;
- to a minicomputer using an IBM/BSC telecommunication terminal emulator software package, i.e. PDP-11 with any operating system, PDP-10 etc.;
- to a communication network supporting IBM 2780.

3.3. Telecommunication capabilities on level 3 and 4 - user level

On the user level the RTS/BSC package can be used to add three different features to PDP-8's, OS/8, RTS/8 and to COS 300 operating systems:

3.3.1. Capabilities in the OS/8 or COS 300 operating system

The package can be used to add telecommunication capability like a new standard peripheral device to the operating system. Telecommunication in the OS/8 looks like a new character oriented input/output enabled multiunit (on a multipoint line) device which can be used from any standard OS/8 program or high-level language like:

- OS/8 FORTRAN
- OS/8 BASIC
- OS/8 FOCAL
- OS/8 MINI-COBOL

In the COS 300 commercial operating system the telecommunication lines are implemented as logical units with sequential access. Several logical units can be used from the high-level language of the COS 300, DIBOL, or can be attached to the Data Entry Package of the COS 300. Telecommunication can be performed simultaneously from DIBOL and from some of the foreground displays. The telecommunication link can be used from the utilities of the operating systems like PIP (Peripheral Interchange Program) or in program packages running under the batch monitor of the operating systems.

The capability to handle the telecommunication link from high-level languages makes it possible to easily implement several high-level telecommunication protocol functions and line-monitoring experiments.

Measurements:

- assembling of telecommunication test packages consisting of messages of different length or with special content. Transmitting and receiving of these packages;
- request for local measurement results from central site, assembling of measurement results into special messages and forwarding them to a control centre, receiving of such packages summarizing and analyzing them on central site, performance of network measurements;
- performance of long-term unattended measurement cycles or special measurements under operator's supervision.

Implementation of level 3 and 4 telecommunication protocols:

- implementation of messages which control the program and operating system running on the destination machine. Transmitting of batch control files;
- building of logical lines over several nodes with fix routing;
- implementation of a routing algorithm;
- implementation of packet switching.

Implementation of high-level functions of telecommunication terminals:

The different features of different types of telecommunications terminals can be easily emulated with high-level language programs. Special capabilities such as data handling, handling of special control characters, data compression/expansion, handling/emulating of special local peripherals such as displays, casettes and performance of background computing can be easily implemented.

3.3.2. Using the telecommunications subsystem in the RTS/8 real-time operating system

The package can also be used in the stand-alone RTS/8. In the RTS/8 the package looks like a standard device handler and is message syntax compatible to the device driver modules of the standard mass-storage devices. The package enables the user to use the telecommunication capability while running real-time tasks dealing with laboratory measurements or data acqusition. The RTS/BSC can be used to connect several PDP-8's and to distribute the real-time work among them.

3.3.3. Using RTS/BSC as a stand-alone package in OS/8 or COS 300

A stand-alone RTS/BSC package enables the user to emulate on his small computer an IBM 2780 or IBM 3780 or SIEMENS Transdata 840X terminal. Use of the stand-alone package under the BATCH monitor, makes it possible to add telecommunication, to any job stream running on PDP-8.

4. BSCMON - THE MEASURING MONITOR PROGRAM FOR RTS/BSC

4.1. The monitor characteristics

Level 1

The program monitors the direction of the telecommunication line, the line turn commands, the line turn completion and the line turn time-outs (for half-duplex modems) as well as the events of sending or receiving of a character or input character time-outs.

Level 2

BSCMON monitors the major state of algorithm (i.e. send or receive), the bit and character synchronisation phases, the characters sent or received and distinguished the control characters from text characters. It traces the telecommunication algorithm on both sides of line: control phase sequences, bid retries, unsuccessful inquiries; sending/receiving of data blocks, parity errors (ASCII data) block check errors, positive or negative block acknowledgements and successful or unsuccessful retransmissions. It shows finally also the algorithm error recovery sequences like acknowledge sequence errors and local or remote abort sequences as well.

Level 3

This level appears in form of external events in the monitoring listing. External events are inquiries for logical connections. Establishing of logical links for send or receive with local logical line number and local user of the link. Arrival of user data request with buffer location and length. Finally the irregular abort or the regular closing of a logical link.

4.2. The method of monitoring

Monitoring is accomplished by several assembler level routines inserted into the level 1 and level 2 assembler code of RTS/BSC. These routines collect the monitored events which are represented by special codes and put these unique codes into a core resident alternating buffer.

An external RTS/8 task (the BSR task) writes one half of the buffer on the mass storage device while the other half is filled with control codes. The file on the mass storage device is used as an endless ring buffer, therefore the last N events are stored in the disk buffer, where N (the length of the disk buffer) is determined by an assembler time parameter. At the end of the telecommunication acitivities a special end-of-buffer marker is written into the file to indicate the beginning/end of the ring buffer.

4.3. Evaluation of RTS/BSC activities

Evaluation occurs off-line because of the speed of the telecommunication line. A BASIC program converts the data in the control buffer and interprets them, currently a line printer listing is produced which contains the last 2 K events. Also some formating is done to make listing easily readable:

- consecutive "SYN" characters are compressed
- the listing is broken up on each line turn and on each end of a telecommunication session
- the control characters are commented
- the events are explained

4.4. Further possibilities

Instead of the listing it is also possible to produce a file containing the compressed information about the telecommunication activities available for any OS/8 FORTRAN or BASIC program for further statistical or analytical evaluation.

5. TELECOMMUNICATION AND NETWORK MEASUREMENTS

For the different measuring experiments the RTS/BSC package can be regarded as the measured and BSCMON as the measuring software, the packages enable us to implement several measuring experiments where the listed parameters can be modified.

5.1. Level 1

- speed of telecommunication line from 600 to 9600 baud
- half-duplex or full-duplex modem/line
- bit synchronisation enable/disable
- number of synchron characters sent/to be received
- time interval between character synchronisation sequences

5.2. Level 2

- package length
- number of packages in a message
- message length
- content of the package (special characters, long sequence of special characters)
- delay between send/receive of packages

5.3. Level 3

- using of multiple logical lines
- different loadings of simultaneously used logical lines
- different lengths of messages for simultaneously used logical lines
- abort of selected logical lines

5.4. Loading of the computer

- modifying of the HW throughput of the computer
- DMA (cycle stealing) loading
- number of external interrupts and software interrupt overhead
- high priority task level load
- background computing load

6. NETWORK SIMULATOR WITH VARIABLE PARAMETERS

6.1. Our objectives

The simulator has two purposes:

- observation of the behaviour of computer networks
- testing of prospective nodes or hosts under a wide variety of shifting conditions.

This means in the first case to establish the problems of the model to be measured and in the second case to devise a measuring system (containing the simulator as signal generator and test load).

We attacked the problem with a simple, highly modular simulator. It is possible to insert a measuring module into this simulator or to insert the simulator itself in a measuring system or to let them work concurrently in a multiprogram environment.

Its chief part is a driver program which can model any traffic network (net of roads, railway, telephone, neural structures, any formation on which the load forms a stochastic process).

The simulated network is produced as a series of discret states. The transitions are observed periodically - the time-unit of the simulator is accordingly a period, composed of 100 tacts.

The driver program accepts some series of signals on which it performs the acts corresponding to forwarding these series. The accepted signals may be given

- by the random number generator of the driver
- by the experimenter (via the teletype console)
- by a host or a node of a real or simulated network (via the appropriate interface).

The present assembly interprets these series as message traveling on a computer network. Accordingly, the meaning of the signals are:

1. identifier of the source machine
2. identifier of the destination machine
3. identifier of the node where the message actually resides
4. data describing the structure and the length of the message
5. the age of the message (i.e. the number of tacts elapsed since its entry into the traffic).

(The last is of course zero at the time of acceptance and is incremented by the simulator during the message's active life.)

As the messages proceed, the state of the simulated system changes; the driver observes and reports the relevant events.

6.2. The driver program

The driver is responsible for

1. the timing of message departure
2. the timing of pass-overs of messages
3. the timing of the break downs of the network's various entities
4. the activating of the segments which carry out the above actions
5. initiating a report on the age and actual residence of each message traveling on the network after every period
6. initiating a snapshot on the state of the network as a whole after a given integer number of periods.

In order to fulfill its duties, the driver program requires the following data:

1. the number of nodes of the simulated network
2. a seed for the random number generator
3. the number of periods after which it has to give the snapshot
4. the functional parameters:
   - the average number of departing messages per period
   - the average number of line breakdowns per period
   - the fraction of faulty deliveries (the pass-over of which has to be repeated)
   - the data concerning the structure of the messages (maximal number of packets per messages - if zero, then variable length is supposed).

These data are indispensable for the proper working of the simulator. Additionally the experimenter may assign two adjacent nodes which are to be disconnected during the first period after their designation. He may suppress the period-reports too.

The simulator request the above data from the experimenter or the test-run operator as follows:

- all of them during the initiating dialogue with the frame
- 2 or 3 after the specified number of periods (this number is given in the 3rd part of the dialogue).

The seed controls the simulation process. If it is positive, the whole dialogue takes place, if zero, the already given parameters prevail, if negative, the simulation process terminates itself. In the latter two cases the 4-th part of the dialogue is skipped, but the snapshot is duly given.

The entries of the messages and the line breakdowns form Poisson processes with stationary increments, with a., resp. b. as parameters for the underlying distributions. The distribution of the faulty (and repeated) passovers is uniform, that of the length of the messages is exponential.

The first question of the dialogue has some far reaching implications.

6.3. The exchangeable units

If the simulated network is ring structured, or serially structured, then the number of nodes is quite enough for the program.

However, if the pattern is more complicated a wide variety of forwarding and routing techniques are possible and it can become desirable to test and analyse any of them. This happens by means of different interchangeable forwarding routines inserted into the frame and called by it whenever necessary.

The driver has no knowledge of the technicalities of the forwarding processes, neither of their strategies, nor of the topology and the technical data of the network.

The forwarding routines may need various (and rather different) information on the network. In this case it is the INPUT programsegment which calls for the network description and puts it into the prearranged tables. The INPUT is the first exchangeable segment; it has to be written according to the needs of the updating and routing strategies.

After the intake of the description through the INPUT segment

- and of the parameters via the 2.-4. parts of the dialogue
- the steps initiated by the driver are executed by the following exchangeable segments
- the segment UPDATE enlists the generated or accepted message into the queue of the first station of its path; this segment arranges the pass-over of the message to the succeeding stations of its propagation - the pass-over process simulates the procedures of the prevailing protocols in a simple way and updates the tables of the program after every event
  - UPDATE is called only by the driverprogram -
- the segment SHUTDOWN actualizes the line breakdowns between one or several pairs of nodes determined either by the experimenter or by the random-generator or both; when a node loses in this way its last living line, the segment notifies the simulator with the registration of its shutdown in the appropriate tables (tables used by UPDATE e.g.)
  - SHUTDOWN is called only by the driverprogram -
- the segment ROUTING (if there exists any routing) handles the tables of the path-generation, it assigns the next station or the whole to the messages
  - ROUTING may be called by UPDATE and SHUTDOWN both but the driverprogram never knows when.

The exchangeable segments are chosen when loading the simulator. They determine fully the mode of the simulation.

The driver activates the snapshot-segment called REPORT. This works on the tables of the whole program, generates its reports and delivers them either on the lineprinter or by spooling on the disk.

7. CONCLUSIONS

As the development of remote data processing systems came to prominent importance, our attention turned to the monitoring of small- and medium-scale computer networks. We worked out a monitor architecture and some basic modules. The highly modular architecture of the described network measuring package enable us to build up a very flexible measuring package being able to execute a wide variety of network measuring experiments.

LITERATURE

1. L.Kleinrock: Queueing Systems Vol. II. pp.422-515
John Wiley & Sons, New York, 1976.

2. G.J.Nutt: Computer System Monitors
Computer 1975. nov. pp. 51-61

3. L.Svobodova: Computer Performance Measurement and Evaluation
Methods: Analysis and Applications
Stanford University, Technical Report No.72

4. BSC General Information
IBM GA 27-3004

5. IBM 3780 Data Communication Terminal
IBM GA 27-3063

6. RTS/8 Users Manual
DEC-08-ORTMA-B-D Digital Equipment Corporation, 1975.

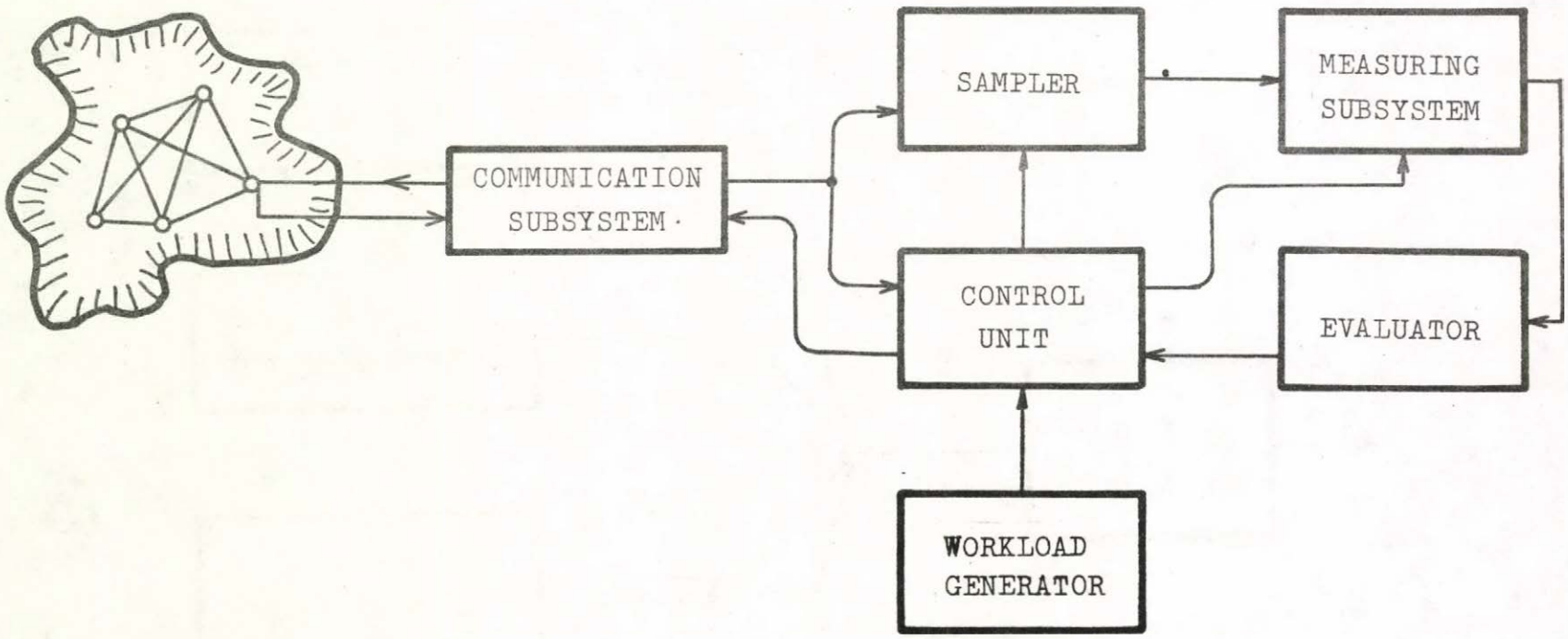


Fig.1 The scheme of monitor

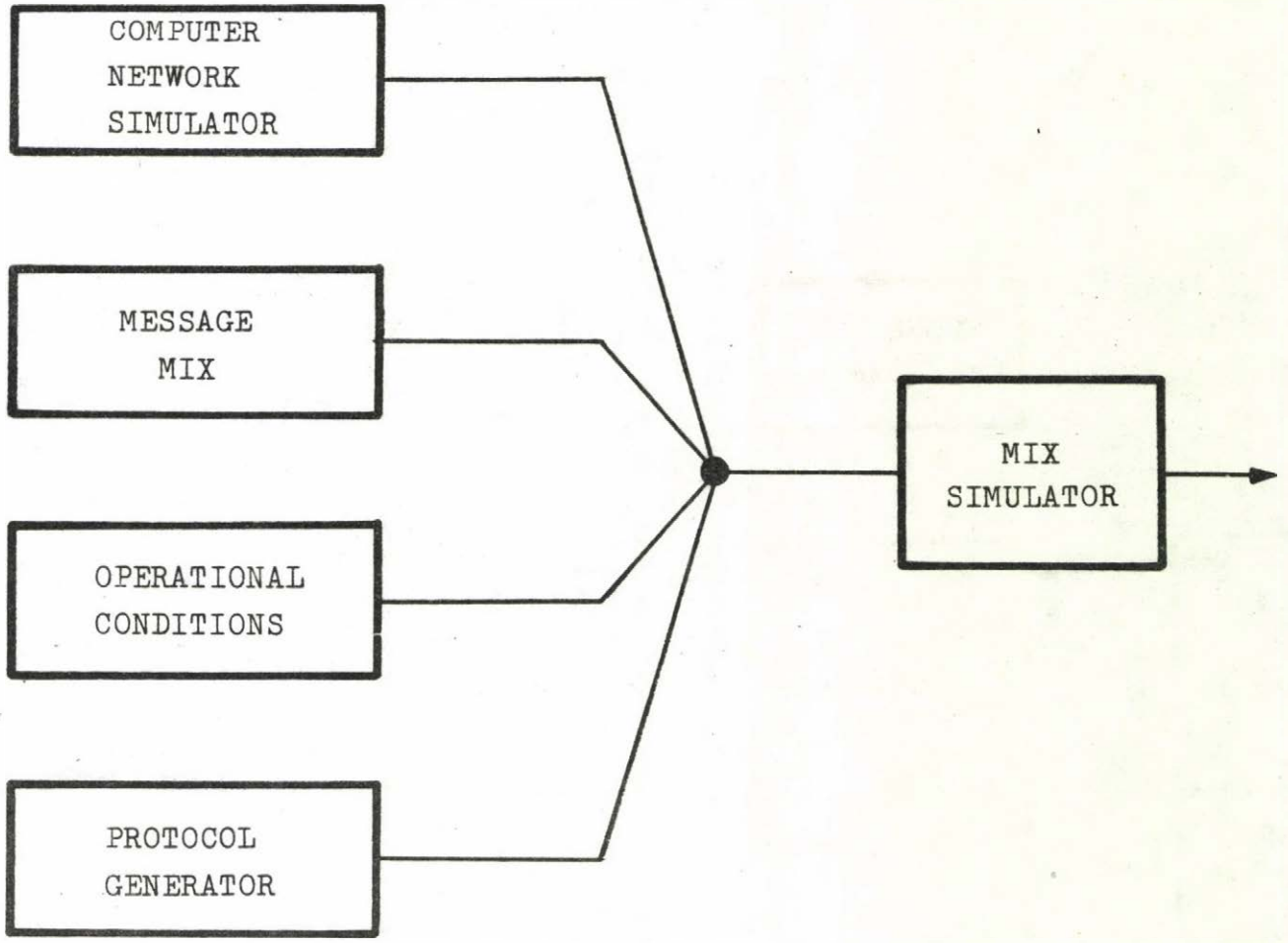


Fig.2 The workload generator

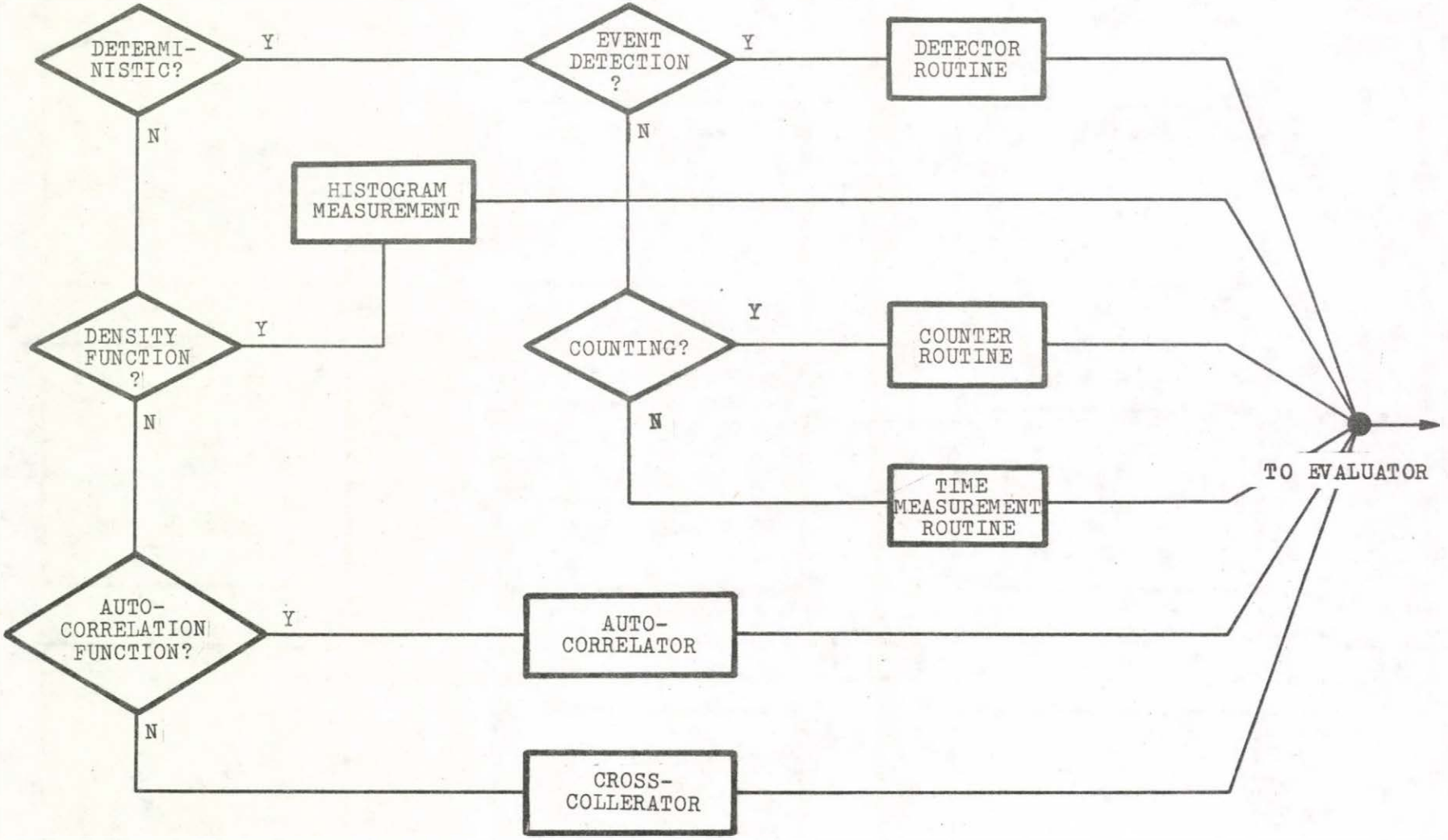


Fig.3 The scheme of measurement subsystem

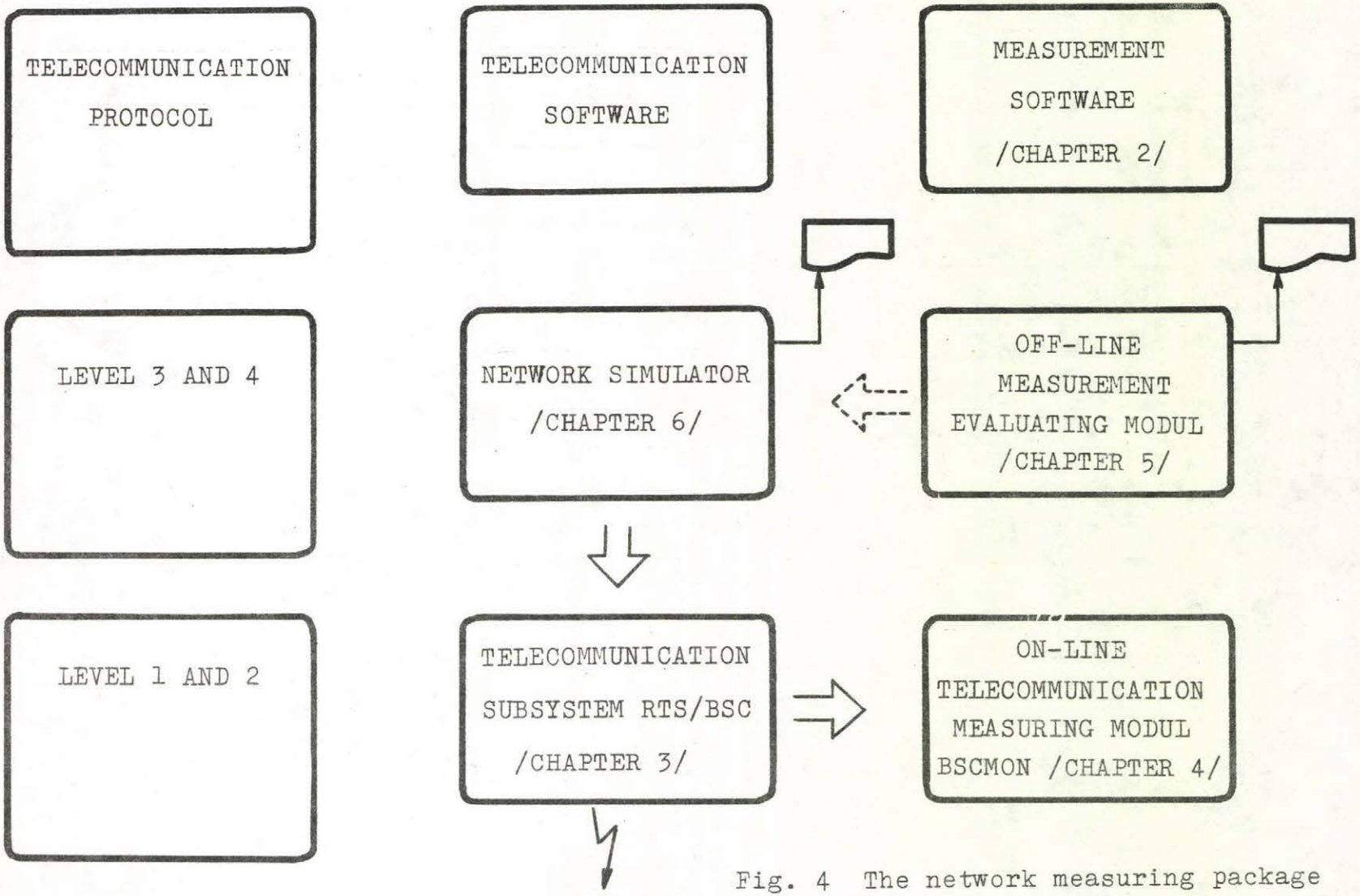


Fig. 4 The network measuring package

AN APPROACH TO PERFORMANCE EVALUATION AND MENAGEMENT OF A TIME-SHARING COMPUTER INSTALATION BY ANALYTICAL METHODS

Zbigniew Błaszczyk
Adam Peterseil
ZETO - Wrocław, Poland

The paper presents a performance evaluation study of a time-sharing computer installation in a commercial computer center by means of analytical methods. The aim of the study was to predict conditions assuring an increment of the system throughput and decrement of job elapsed time within a given cost constraints.

The performed study has shown that the application of even relatively simple analytical methods to the evaluation of the complicated computer system without a considerable decrement of the accuracy of results is possible. Certainly, there were some discrapances between the real system and the simple model which was used. That is why the insertion of the model modification was necessary.

The advantages of such an approach to the computer system performance evaluation are its low cost and a short study duration.

Introduction

The basic measures of a computer installation quality in a commercial computer center are throughput and elapsed time of jobs. Elapsed time is particulary important in the case of mixed batch and on-line system workload.

The bigger throughput allows for more users in a region to be served and can be achieved by an increment of a computer system utilisation, so by bigger resources loadings.

The resourse "i" loading is defined as

$$\rho_i = \frac{\text{time whed theresource "i" is busy}}{\text{time when theresource "i" is available}}$$

Before starting the performance evaluations study it was found that both throughput and elapsed time are not good. The most money consuming resources were too little loaded and the on-line users were not satisfied because of too long waiting time.

The possible reasons were:

- bad computer operators service,
- bad installation parameters values for the given system workload [BAPE],
- low system reliability,
- unfitness of the system configuration to the system workload.

On the basis of 3 month monitoring by means of the standard ICL Performance Package, the System Journal and by

own designed monitoring programs it was found that the primary reason of the bad quality measures was the inappropriate configuration to the system workload. There were the significant system bottle-nacks.

So it was clearly seen that the evaluation problem is as follows:

- what is the way of changing the computer configuration in order to achieve better quality factors?

The constraints were:

- it was possible to change peripherals only,
- the study / including data analysis achieved by system monitoring/ was to be finished during one month,
- no more than 0.2% of the computer installation cost could be spent for the study.

The method description.

The configuration of the evaluated computer system based on the ODRA 1305 c.p.u. / correspondent to the ICL 1906/ is shown on the fig. 1. The system is operating under control of the general purpose ICL GEORGE-3 operating system /more details one can find in [GE01] and [GE02] references/.

The basic evaluation tool was the FAST technique [ICLT]. For the aim of the paper the following topices of the FAST are important:

1. there are N independent devices in the model,
2. overlap factor $OV = \sum_{i=1}^{N} \varrho_i$ is assumed to be the performance measure,

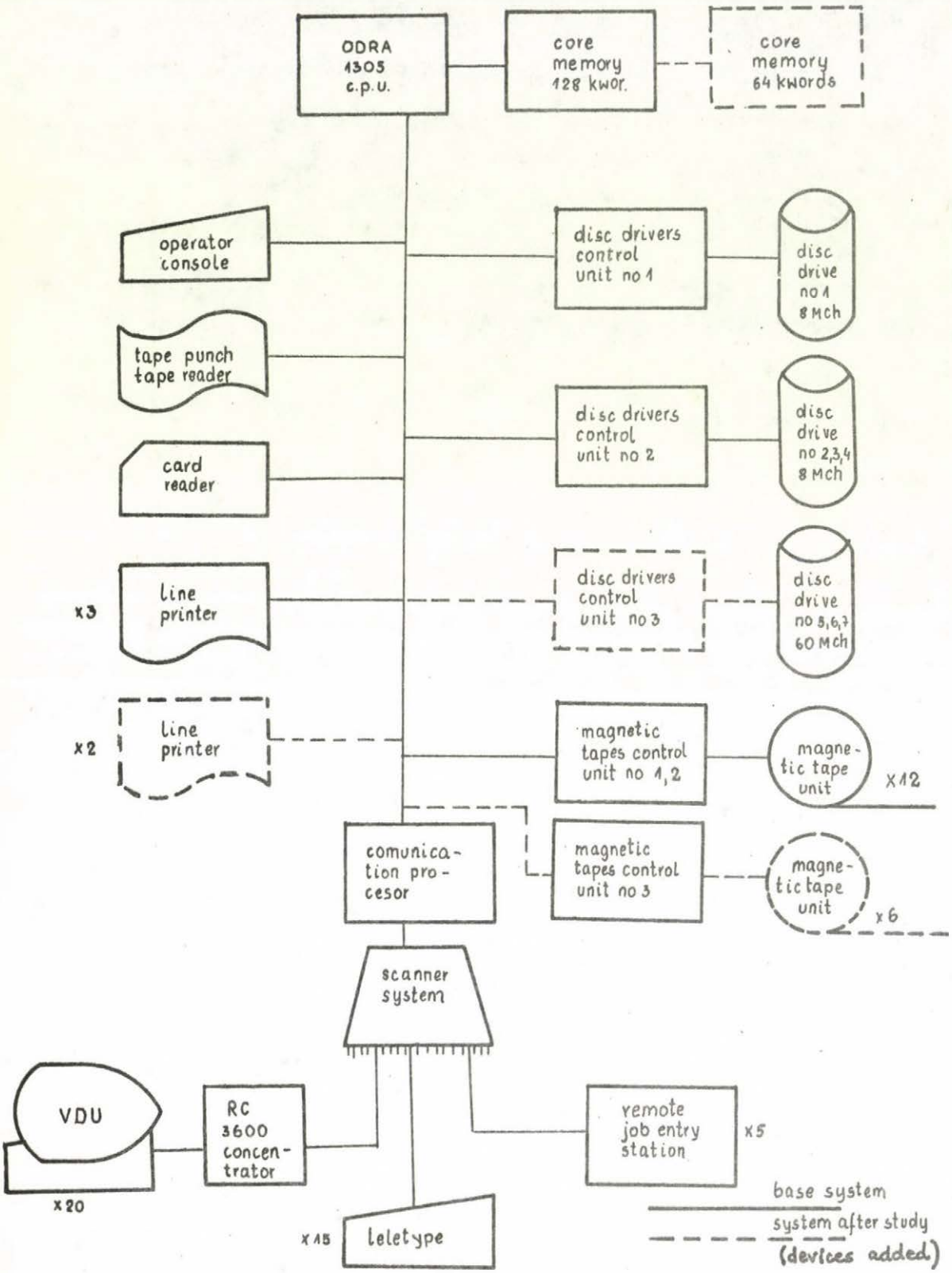


Fig.1 The computer system configuration.

3. a process "p" using the specified device at any given time instant is the basic workload component,
4. a program can generate more than one process,
5. a computer system load is characterized by the

$$\frac{\text{number of programs}}{\text{number of concurrent processes}} \quad \left(\frac{\text{program}}{\text{process}}\right) \text{ ratio}$$

and this ratio for the given workload is constant,

6. a measurment results of the existing resources loadings are the input data to FAST,
7. the model calculates the value of

$$OV = F(p) = \frac{S_{p-1}}{S_p} \quad , \quad S_p = \sum_{i=1}^{N} \left(A_p(N,i)\right)$$

and

$$\left[A_p\right] = \begin{bmatrix} X_1 & 0 & 0 & \dots & 0 \\ X_1 & X_2 & 0 & \dots & 0 \\ \vdots & & & & \\ X_1 & X_2 & X_3 & \cdots & X_N \end{bmatrix}^p$$

where:

p - denotes the number of processes $(p \in \mathcal{N})$

$x_i = \frac{S_i}{OV}$ - denotes the reduced resource "i" loading,

$\mathcal{N}$ - denote integer numbers,

8. the increment of the number of processes causes the equal increment of all reduced resources loadings with the

$$\frac{OV_{\text{new system}}}{OV_{\text{old system}}} \quad \text{ratio}$$

9. the number of processes increase proportionally to the

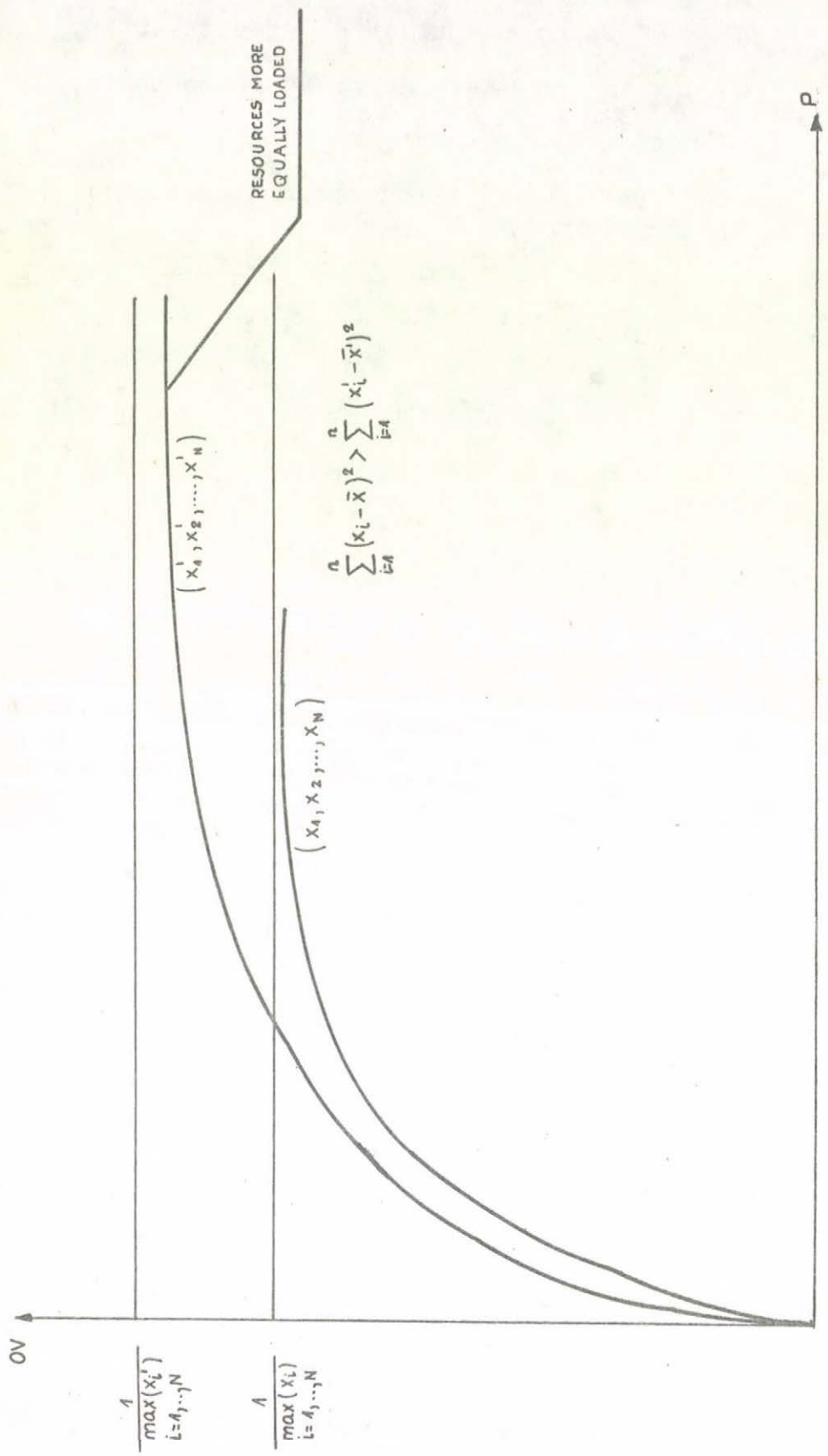


Fig. 2 The typcal shape of the FAST curve.

increment of the core memory available for users programs, i.e. with the

$$P = \frac{\text{core memory available for users programs}}{\text{mean core required per one program}} \times \frac{\text{process}}{\text{program}} \text{ ratio}$$

10. the model is based on the theory of the cyclic quening systems, i.e. that the finite number of servers in a closed loop is assumed.

Fig. 2 shows a typical Fast curve.

The assumptions 2,8,9,10 were not fulfilled in our considerations.

There were the following reasons discussed with the way to omit them:

1. there is the filestore in the system which is held on the disc storage /on-line filestore/ and on magnetic tapes/ rest of the filestore for which there is not enough space on the disc/.

When a program can not find its own file on a disc, the operating system retrieves this file from appropriate tape. File retrieving takes a long time during which the magnetic tape unit and the control unit are busy. That is why an increment of disc storage capacity causes unequal increment of all reduced resource loadings.

In a particular case, an increment of the disc storage capacity causes decrement of magnetic tapes units loadings /retrieves happen only occasionally/ and increment of all other system components.

FAST does not take into account the resources capacity attribute explicitely /it takes into account the resource speed only/.

During the evaluation study an empirical dependence between magnetic tape units loading and the capacity of the on-line filestore achieved by means of the frequency of the accesses to the filestore monitoring, was used.

2. the operating system GEORGE-3 has its own special core memory administration strategy. Just like above an empirical dependence between the core for users jobs and allocated by operating system, was used.

   On the base of the measurments the following empirical formula for the number of processes as the function of the core memory size instead of mentioned in item 9 for FAST, was found:

$$P_{\text{new sys.}} = \frac{\text{core memory size} - 40.5 \text{ kwords}}{\text{mean core per program} + 4.6 \text{ kwords}} \times \frac{\text{process}}{\text{program}}$$

3. the number of jobs in the system was not fixed in fact. Also each. job had its own parameters / c.p.u. time, disc store transfers, memory required etc. /.

   For the purpose of the analysis, 3 month meaasurment of the workload was made by means of own designed monitoring program.

   Figs 3 shows the exemplary measurment results. It was found that every day workload characteristics vere steady anough to treat them as the fixed, so the technique could be applied.

4. the overlap factor OV /see item 2 for FAST/ could not be used as the performance measure when the resources speed was changed.

That is why the relative system throughput RST [DRUM] was used instead of the OV. It may be expressed by the formula:

$$RST = \frac{T_b}{T_n}, \text{ where}$$

T_b - is the computation time of the given workload on the base system,

---

Job statistics for the period 3-24.01.

achieved by the program analysis of the SYSTEM JOURNAL FILE.

Sample size.

- total system active time	370979 secs
- total amonnt of processed jobs	1748
- amount of jobs processed per hour	16.92

Job statistics.

- mean elapsed time-c.p.u. time	1797.438	dispertion	3076.430
- mean c.p.u. time	35.019	"	185.191
- memory required	5.468	"	8.391
- c.p.u. time per program	19.981	"	199.278
- off-line transfers/to the filestore/	2201.681	"	13071.876
- on line transfers /to MT/	945.786	"	8069.955

All times are in seconds, memory in kwords /1 word= 24 bits/, transfers in thounsands.

Fig. 3 The exemplary results of the workload measurment.

T_n - is the computation time the same workload on the new system.

There were two cases of the evaluation process:

case 1 when the number of system components was changed without changing the speed, then

$$RST = \frac{OV_n}{OV_b},$$

case 2 when the system components speed was changed, then

$$RST = \frac{T_b}{T_n}$$

The processing time was predicted from the elapsed time /calculated by means of sequential M/M/N model /and the mean level of multiprogramming /with fixed $\frac{\text{number of processes}}{\text{number of programs}}$ ratio/.

The evaluation process.

The results of the system /base/ monitoring are gathered below:

1. the components loadings
   - central processor unit - 0,45
   - disc drives control unit No 1 - 0.20
   - disc drive No 1 - 0.60
   - disc drives control unit No 2 - 0.19
   - disc drive No 2 - 0.42
   - disc drive No 3 - 0.48
   - disc drive No 4 - 0.44

- magnetic tapes control unit - 0.825 each
- line printers - 0.19 each
- card readers - 0.10 each

2. the overlap factor OV_b= 5.11,
3. the mean job elapsed time t_e=1820 secs,
4. the mean job service time s = 980 secs, so $t_e - s = 840$ secs
5. one-day jobs stream processing time T_b = 484 mins,
6. $\frac{\text{number of processes}}{\text{number of programs}}$ = 1.1499,
7. the mean core memory size per program - 5.51 kwords.

The following conclusions were found:

1. too little c.p.u. loading,
2. good filestore allocation on the disc drives No 2,3,4 /the drives were equally loaded/,
3. high disc drive No 1 loading comes from the operating system overlays transfers,
4. high MT subsystem loading comes from too often file retrives,
5. bad $\frac{\text{job waiting time}}{\text{job service time}}$ ratio.

Many trials with changing the configuration were made. Finally we found that within the specified constraints the optimal configuration is as shown on the fig. 1 /detted lines denote the added devices/.

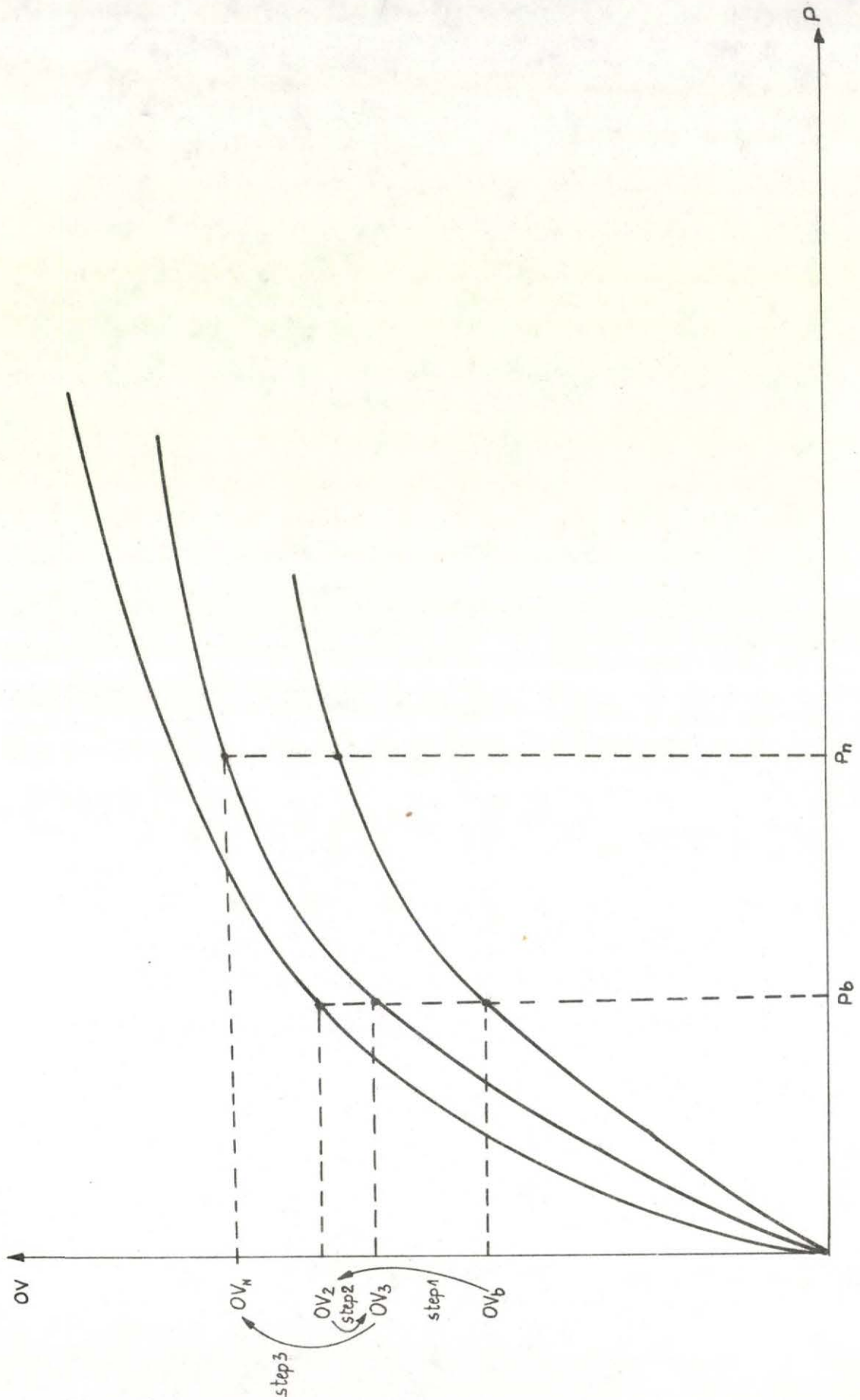


Fig. 4. Steps in the evaluation study.

The predicted new system parameters was as below:

1. the components loadings:
   - central processor unit - 0.74
   - disc drives control unit No 1 - 0.273
   - disc drive No 1 - 0.83
   - disc drives control unit No 2 - 0.20
   - disc drive No 2 - 0.30
   - disc drive No 3 - 0.30
   - disc drive No 4 - 0.30
   - disc drives control unit No 3 - 0.288
   - disc drive No 5 - 0.38
   - disc drive No 6 - 0.435
   - disc drive No 7 - 0.399
   - MT control units - 0.439 each
   - line printers - 0.258 each
   - card readers - 0.20 each
2. the overlap factor $OV_n = 7.136$,
3. mean job elapsed time $t_e = 1077$ secs,
4. mean job service $s = 600.7$ secs, so the waiting time $t_e - s = 476.4$ /note that new job waiting time is almost twice shorter than in the base system/,
5. one day job stream processing time in the new system $T_n = 251$ mins.

The above results were achieved in four steps. Three of them were concerned with system throughput /see fig. 4/, the last one with the elapsed time:

step 1. The additional magnetic tape control units, external stores and line printer allowed for achieving more equal components loading, thus bigger overlap factor /OV_2/.
The increment of the system throughput was

$$RST_1 = \frac{OV_2}{OV_b}$$

/Note that for the calculation of magnetic tape control units loading the empirical dependence for the filestore was used/.

step. 2. The additional disc drives from step 1 were changed to the faster ones / 60 Mch /. The overlap factor / OV_3/ was decreased, but the system throughput was increased:

$$RST_2 = \frac{T_b}{T_2}$$, where T_2 denotes the workload processing time with new disc drives.

step 3. The increment of the core memory size allowed for the number of processes-and thus the overlap factor-increment /note that the empirical formula for "p" was used /:

$$RST_3 = \frac{OV_n}{OV_3}$$

The total system throughput increment was:

$$RST = RST_1 \times RST_2 \times RST_3 = 1.93$$

From above and the RST definition

$$T_n = \frac{T_b}{RST} = \frac{484 \text{ mins}}{1.93} = 251 \text{ mins.}$$

step 4. On the base of new system components loading and workload analysis the job elapsed time and service time by means of queuing theory were calculated .

The additional comments are needed to the following points:

- the smaller service time comes from the much faster disc drives and the decrement of the number of retrieves,
- the decrement of the MT subsystem loading comes from the bigger filestore size.

Conclusions.

The effect of described study was the increment of the throughput in 93%. The elapsed and waiting timeswere twice lowered. The results were tested on the reconfigured installation. We found the equivalence with the predicted values within 15%. It seems to be accurate enough especially taking into account the simplicity of the model, cost and the duration of the study.

Bibliography.

[BAPE] Bazewicz M., Peterseil A., An approach to adapting multiaccess time-sharing system to users requirements by simulation methods, paper presented on the II International Workshop on Modelling and Performance Evaluation of Computer Systems, Stresa, Italy, 1976

[BLAS] Błaszczyk Z., The evaluation of computer system for a commercial computer center by means of analitical method, ZETO Information Bulletin, No 1/78, in Polish

[DRUM] Drummond M.P., Evaluation and Measurment Techniques for Digital Computer Systems

[GILB] Gilb T., Datametrics Handbook, Control Data Institut, 1974, manuscript

[GEO1] Operating Systems GEORGE-3 and 4, ICL, London, 1972

[GEO2] GEORGE-3 and 4 Operation Menagement, ICL, London, 1972

[ICLT] Practical Sizing, ICL Training and Education Publication, 1975

A FREE MEMORY ORGANIZATION FOR AN OPERATING SYSTEM OR A FRONT-END PROCESSOR

T. Muehleisen - W. Wietrzych

The Technical University of Wroclaw

In the paper an organization of free memory designed especially for an operating system of a front-end processor is presented. A brief review of problems involved is given and concluded with the introduction to the free memory structure and a memory remarks on the practical memory implementation and hitherto experiences are presented.

CONTENTS

I. INTRODUCTION

In the recent years it has become a common practice in multi-access computer systems to transfer telecommunication functions from the main computer to a dedicated machine refered to as communication processor or front-end processor [1,2]. Such off-loading of the main processor causes a considerable increase in the overall system processing power resulting in better performance/cost parameters for the whole system.

As far as the batch-interactive GEORGE-3 operating system is concerned the throughput improvement can be achieved by using the ICL 7903 communication processor. No-availability of the processor on the domestic market one one hand, and considerable and still growing number of GEORGE-3 system users in our country one the other hand have been two major reasons underlying an idea to emulate the 7903 processor on a computer suitable for the purpose and available on the market. The Polish ODRA-1325 computer has been chosen functionally equivalent to a medium-size minicomputer. The source ICL literature far from being complete and totally different organization of the two processors /i.e. ICL-7903 and ODRA-1325/ compelled the designers to look for unique solutions. The resultant operating system is functionally similiar to its English counterpart but structurally quite different.

To design an appropriate free memory organization for the system was one of many problems to be solved. The memory had to be structured in such a way as to allow for efficient data handling between the host operating system and remote terminals of TTY-7071 type. The framework of the memory structure should make execution of tests possible and some rules of memory management had to be developed.

The paper presents considerations which have led to a free memory structure and makes an introduction to this structure. The problem of memory overflow control has also been solved and the solution presented here-in along with memory management rules. The algorithms have been developed in such

a way as to allow for easy modifications of such memory parameters as: cell size, chain length, number of chains etc. This organization general to some degree makes the free memory suitable for other operating systems of this kind.

The operating system is operational now and observations made through a built-in monitoring mechanism have led to conclusions affirmative in general in relation to the designed free memory organization. A more quantitive analysis of the memory behaviour under normal and abnormal system operation will be a subject of further studies.

II. GENERAL CONSIDERATIONS

The basic functions of any operating system for a communication processor can be summerized as follows:

1/ message buffering,
2/ message formatting,
3/ message editing,
4/ controlling remote devices like terminals and batch-stations via multiplexor or scaner,
5/ communicating with host operating system,
6/ communicating with operator,
7/ making possible execution of testing programs.

The operating system performs all of its functions executing in the core and for this reason an adequate memory organization is essential for the proper system operation. Generally speaking the memory can be assigned to programs, data and to serve as working areas. The program can be an executive routine or a routine to be run under its control. As for the data a distinction should be made between areas for the data permanently resident in the core, such as program valuables, control tables, description tables, etc. and areas for the data residing in the core temporarily. Such data prevails in communication operating systems and it pas-

ses through the memory on its way from the host operating system to remote terminals or in the opposite direction. Thus for one direction of transmission the host computer and the terminals can be considered as the data source and the data sink, respectively /see Fig. 1/. The data appears

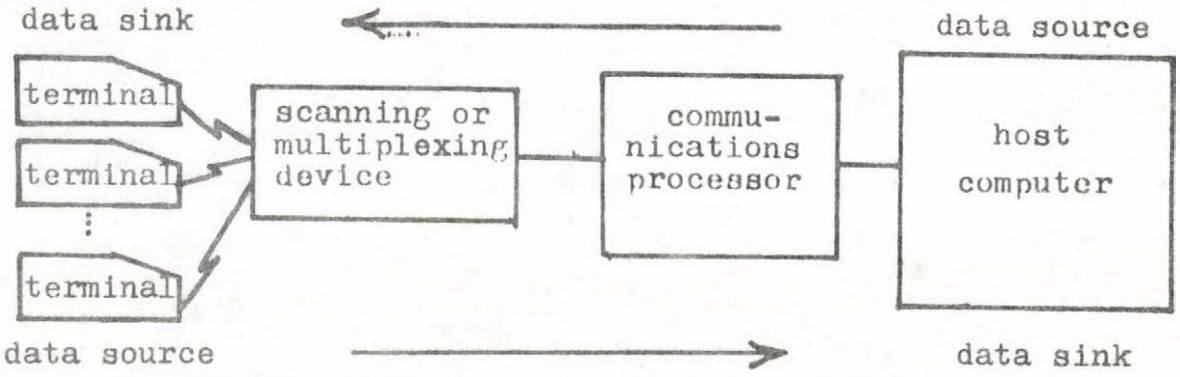


Fig. 1. Data source and data sink in full communication configuration

at the source in random fashion, and for this reason the memory of the communication processor should be assigned to it dynamically. This approach contrary to the fixed memory allocation strategy helps achieve better memory utilization parameters which is especially important in systems with a limited core area. In systems with the dynamical memory allocation a seperate management module is present and it is responsible for all memory requests and returns. In such systems a part of the core forms a dynamical memory pool, also refered to as free memory. The free memory is devided into blocks of equal or unequal size. The management routines keep record of all free blocks and assign it on the on-demand basis. If the memory blocks are equal they form a queue also defined as a chain, where the current block contains the address of the next block. In this case the first block of the chain will be given first. If the memory blocks are not equal all the block addresses and block sizes can be stored in a table for the informa-

tion of the management routines.

Once a memory block has been given to a process the data is input to it. The transfer of data between different memory locations /usually between different queues/ is achieved by attaching the first address of the block to appropriate queues, rather than actual transfering of data. This time-saving approach is widely used no matter how simple the system is.

In the operating system for a front-end processor the data can be information messages or supervisory messages and they have to be transferred between buffers to/from the host computer and terminal queues or in the opposite direction /see Fig. 2/. Terminal queues are used as a buffer of

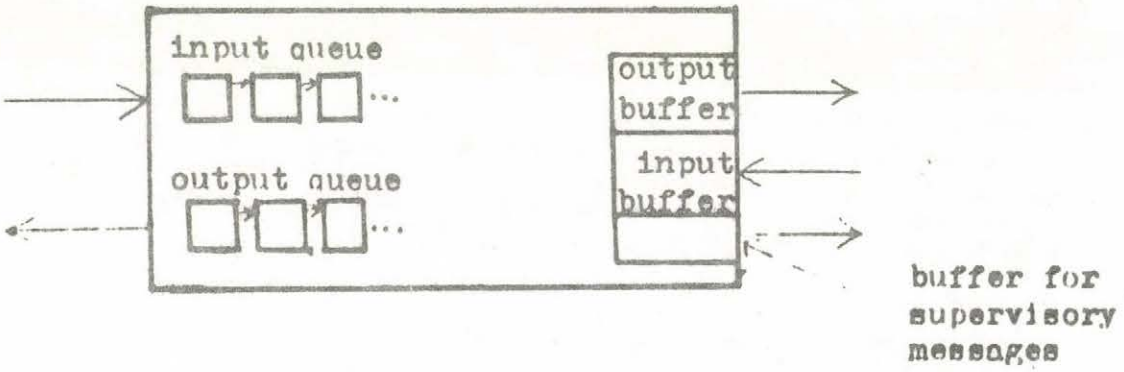


Fig. 2. Extreme destinations for data transfers in FEP

a variable length. The size of a single queue cell should be equal to the terminal message length and this determines the memory unit. A problem may arise when this unit is too big for carrying control data also known as supervisory messages. The possible solution to this problem may be creating a seperate pool of cells for supervisory messages only managed by seperate routines. This problem is non-existent in the system to be designed because according to the ICL standards [6] the supervisory message length can vary between 2 words and 128 words. Thus the same cell can be used for transfering of data and supervisory messages.

III. PRACTICAL FREE MEMORY STRUCTURE

In order to design a practical free memory structure several properties of the specyfic operating system under development must be taken into account. For the communication system in question the following parameters have greatest impact on the ultimate form of memory organization:

1/ lengths of a single message buffered in the FEP,
2/ maximum and average lengths of control data taking the form of supervisory messages,
3/ anticipated size of programs to be run under control of the operating system.

In addition, given the operating system size, the free memory area will depend on the amount of core available to the system. Likewise coding of algorithms will depend on the computer addressing scheme /i.e. whether byte- or word-oriented/.

The analysis of all the factors involved has led to a practical solution, the outline of which will be presented here-in. The detailed description of the free memory structure is given in [4].

The free memory structure is based upon a table refered to as free memory state table /FMST/ or just state table and upon five uni-directional lists of memory cells. The lists also known as chains are totally independant and according to the rule [3] the first word of each cell points to the next cell, the only exception being the last cell, where the control word used for referencing takes the value 0. The first cell of each chain is pointed to by a reference word in the state table. The memory chains structured as above allow for last-in first-out /LIFO/ access strategy only, i.e. the cell currently taken is the last cell returned to the chain. Only the first cell in the chain can be given out and its address can be found in the reference word of the state table. When the cell is returned to the chain it is placed „between" the state table and the cell referenced by the control word for this chain.

The single cell is a free memory unit and its structure

is presented in Fig. 3. The cell length or parameter B has been chosen to be 36 words. The reason for this is the word-oriented addressing scheme of ODRA-1325 and the message si-

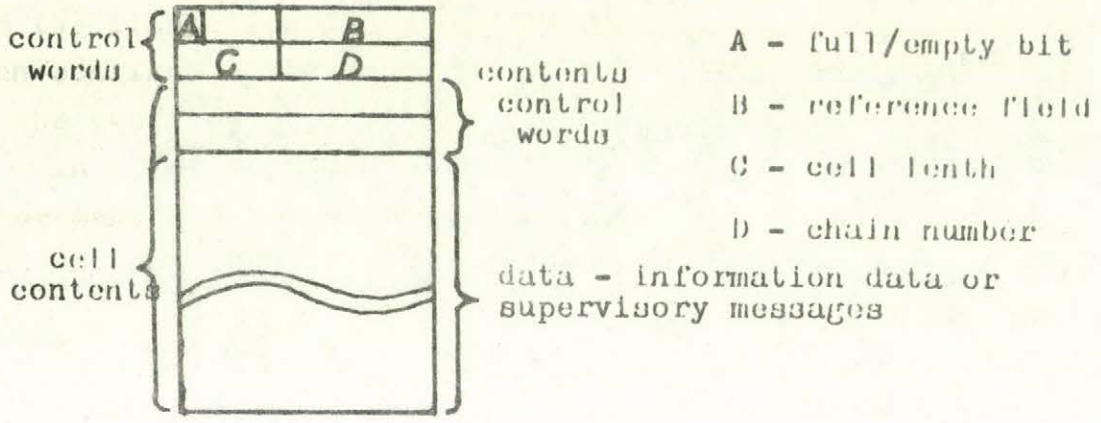


Fig. 3. Cell structure

ze to/from TTY-7071 terminals equal to 128 characters /or 32 words in ODRA/ on top of which 4 words have to be added for control. Once the cell has been taken off the chain its 2 first control words may not be destroyed, except for the reference field which is updated when the cell is returned to the chain it has come from. The remaining 34 words are available to the process which has requested the cell.

One full chain /i.e. with all its cells present/ forms so called free memory block which, if necessery, can be given as a whole. The memory area made available in this way can by assigned to a test or other program to be run under control of the operating system. This involves the total destruction of the chain organization. Upon returning of the block the chain would have to be restored and its cells made available again.

Each terminal and device should be tested by a seperate test. The tests /not written yet/ will be of various sizes and only one can execute in the memory at a time. For this reason the tests can be assigned the memory area in multiples of 3600 words equivalent to 100 cells. The block size

equal to 3600 words has been chosen because the anticipated smallest test size was about 3-3.5 kwords.

The state table and its contents are shown in Fig. 4. The table entries correspond to a situation when a number of cells is being used by some process and one chain has been made empty.

FMST	398	current number of cells
FMST+1	3	current number of blocks
FMST+2	BASE	memory starting address
.	5	total number of blocks
.	0	reference to 1st chain
.	0	number of cells
	BASE + 3672	reference to 2nd chain
	98	number of cells
	BASE + 7200	.
	100	.
	BASE + 10800	.
	100	
FMST+13	BASE + 14400	reference to 5th chain
FMST+13	100	number of cells

Fig. 4. State table entries in normal system operation

The possibility to increment the number of chains is only limited by the core area made available. In ODRA-1325 the core is 32k allowing for 18000 free memory words. The starting address BASE is generated upon creating of system tables and system buffers and it is only then that the free memory can be formed. With all the chains generated /see Fig. 5/ the operating system is ready for operation. Once the normal system operation is started the presented cell order /i.e. consecutive references higher by 36/ is scattered because of random demands and returns of cells to/from the state table. This observation has been used in determining maximum numbers of cells within each chain requested throughout the operation.

The free memory as presented in Fig. 5 and a set of ma-

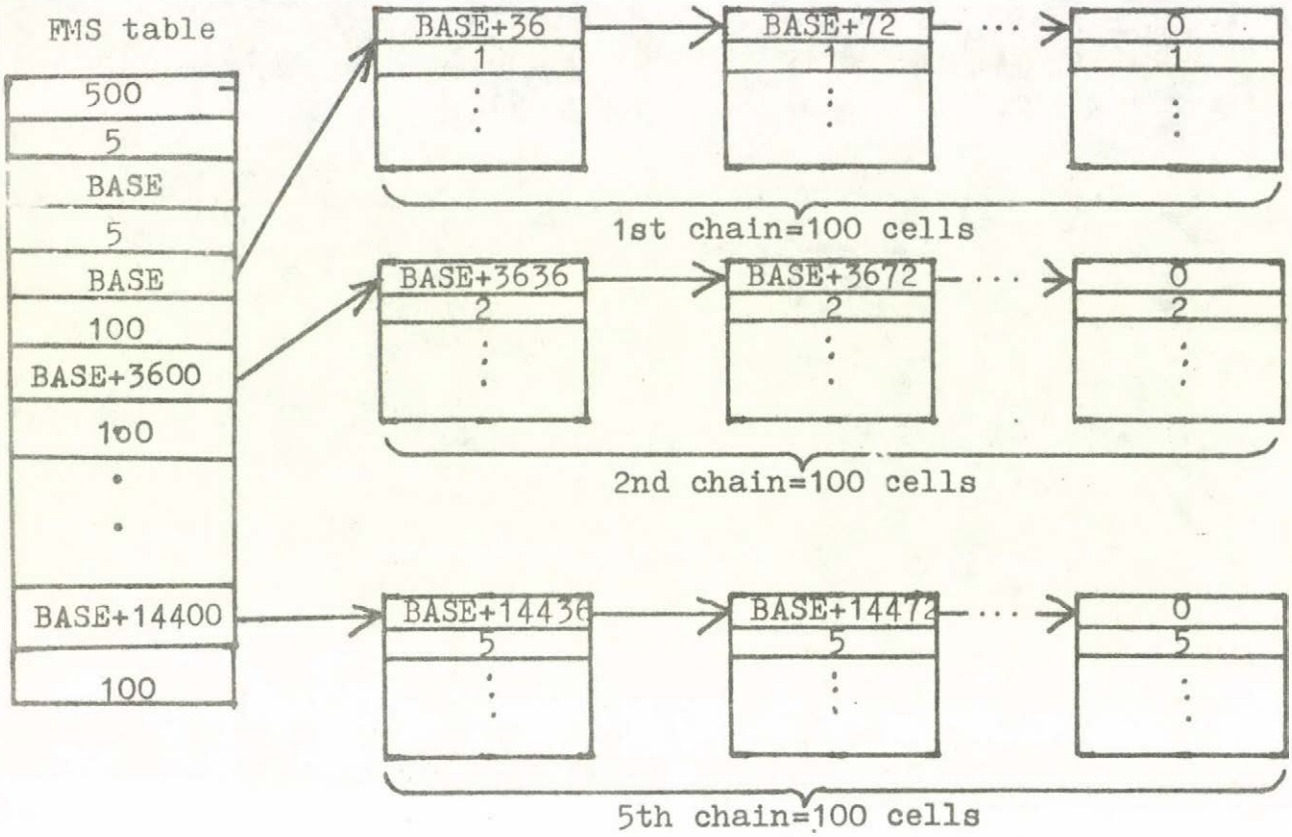


Fig. 5. Complete free memory following formation process

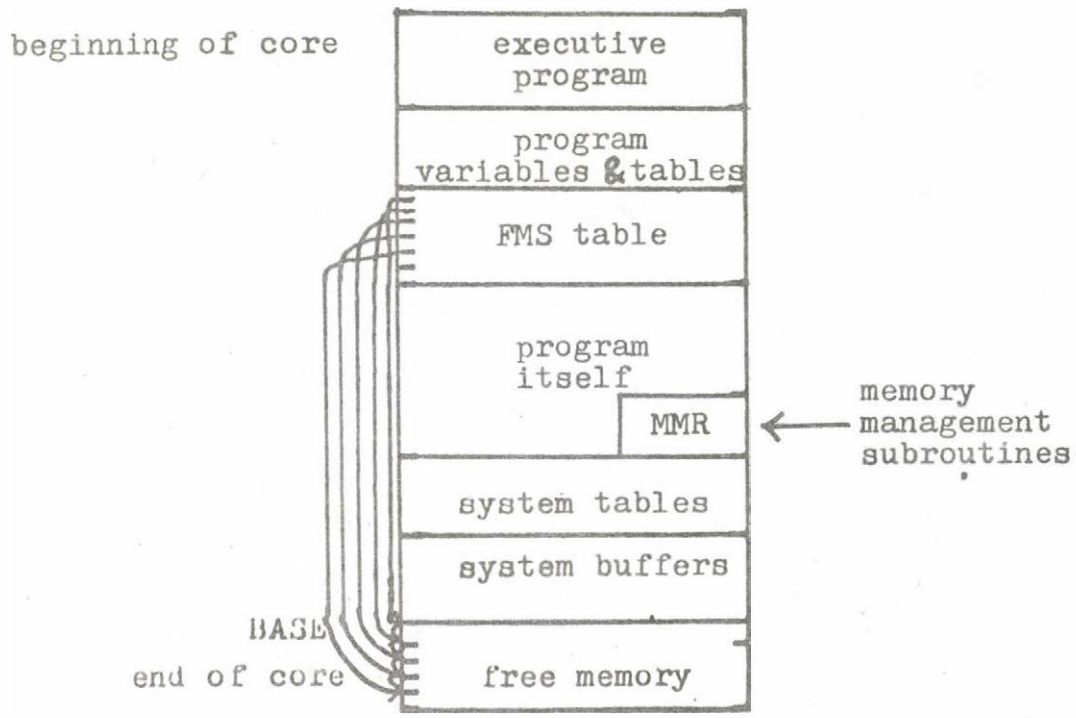


Fig. 6. Practical location of free memory in core

nagement rules form a part of the operating system. Fig. 6 shows core locations of particular components of the system. In case there remains any core area it can be assigned to another program to execute in multi-programming environment controlled by the executive. In ODRA-1325 the core is equal to 32k words making the additional core area non-existent.

Memory management routines depicted in Fig. 6 as MMR are a number of subroutines called upon by processes requesting a cell, returning a cell, requesting a block /or a couple of blocks/ or returning it back. For the sake of the role they play in the operating system the management routines have been assigned high priority. The management routines will be discussed in more detail in the chapter V.

IV. SYSTEM OVERFLOW CONTROL

In the normal system operation all data transfers are carried out in the memory cells and for this reason cells have to be requested by an appropriate process beforehand. Once the data has reached its destination, i.e. buffers to/from terminals and to/from the host processor or the data-lost condition has appeared the cells are released, in one-by-one fashion, and returned to the state table. If throughout the system operation the cell request rate exceeds the cell release rate an overflow condition may occur resulting in the system „breakdown. In order to avoid this undesirable situation a mechanism controlling excessive demands for free memory cells has been introduced.

Procedures of this mechanism cooperate closely with the GEORGE-3 system in adjusting amount of data sent to the front-end processor to the current system requirements [6]. For this purpose 6 overflow and 6 continue levels have been defined. The levels correspond to some numbers of cells in the free memory. Should a cell be taken resulting in exceeding an overflow level an appropriete supervisory message SYSTEM OVERLOAD

will be sent to the host operating system to request it to limit the amount of data sent out. As a result a number of cells will be returned and when returned to the free memory, in one-by-one fashion, a continue level will be reached. A SYSTEM CONTINUE supervisory message will be sent to the host computer and the limitation imposed previously will be lifted. Thus each continue supervisory message cancels conditions set forth by the overflow supervisory message of the same level.

In order to avoid alternating sending of overload and continue messages some histeresis must be introduced in the overload and continue levels, as shown in Fig. 7.

	number of cells	
	overload condition	continue condition
level 1	23	25
level 2	18	20
level 3	13	15
level 4	8	10
level 5	3	5
level 6	1	1

Fig. 7. Overload and continue levels

At the beginning the levels have been defined arbitrarily and then tuned to the above values in the actual implementation phase. The tuning has been based upon conclusions drawn from the observations of the free memory behaviour in overload conditions through a built-in monitoring mechanism. The devised control mechanism gives the process requesting a cell possibility to define how important is the cell to the process. This is achieved by complementing the cell request command GIVEC with a parameter n to inform an appropriate management routine that the overflow level n can not be reached. Thus in overload conditions the cell will only be given if the level n-1 or lower has been reached; otherwise the cell will not be given and the process will recieve a negative answer.

In order to recieve a cell in emergency, i.e. when the

6th overload level has been reached and a supervisory message must be sent, a GIVEC(7) command can be issued. In this case the last cell will be given to the process which has requested the cell. This brings the system to the brink of the break-down condition and should be avoided. On the other extreme a GIVEC(1) command issued in overload conditions will always be answered with a negative reply.

The reference [4] gives a detailed discussion of the system overload control mechanism.

V. MEMORY MANAGEMENT RULES

The free memory is managed by a number of routines allowing for the following operations:

1/ generation of the free memory,
2/ requesting a cell from the state table,
3/ returning a cell to the state table,
4/ requesting a block or a number of blocks,
5/ returning a block or a number of blocks.

The free memory generation takes place after the system tables /so called specification tables/ and the system buffers have been created. At this point the address BASE is defined and input to the memory generation routine. As a result the free memory chains are generated, the first one starting from BASE and the state table is filled with proper entries. At this stage the system is ready for the normal operation.

The memory management routines executing operations 2 through 5 as mentioned above are accessible to appropriate processes by the following commands:

1/ GIVEC(n)
2/ TAKEC(a)
3/ GIVEB(m)
4/ TAKEB(b)

where parameters n,a,m,b stand for:

n - overflow level not to be exceeded

a - starting address of returned cell
m - number of blocks requested
b - starting address of all blocks returned.
The commands and the routines will be introduced briefly here-in. The detailed discussion of all management problems is presented in [4].

GIVEC(n) command may be issued each time a process requests a cell. The routine called upon will look for a first free cell starting from the top of the state table and give out the cell unconditionally if no overflow level has been reached. The process recieves the cell by simply getting informed of the starting cell address. In overload conditions, i.e. when at least the first overflow level has been reached the cell can only be given to the process on condition that it is less than n. Otherwise the process will get a negative reply. Should a cell be given out causing an overflow level to be reached another cell will be requested to carry the SYSTEM OVERLOAD message to a buffer from which it will be sent across the inter-processor buffer to the host operating system.

TAKEC(a) command may be issued each time a process releases a cell with the starting address a and returns it to the state table. The routine called upon places the cell on the chain it has come from. The cell is taken unconditionally, except for the case when a system continue level has been reached. The cell being returned is then given out in order to carry the SYSTEM CONTINUE message to a buffer, from which it will be sent across the inter-processor buffer to the host operating system.

GIVEB(m) command may be issued in every case an area is needed for a test or other program to be run under control of the operating system. Since one block is equivalent to one full chain and the last chains in the free memory tend to remain untouched, blocks will be given starting from the bottom of the state table. The process will get the area of m blocks only if at least m last chains are full. Otherwise

a negative reply will be given. Only last contiguous blocks can be given out and the requesting process is informed of it by the first block address it recieves. The released area extends between this address and the last address of the core. The parameter m allows for adjusting the area to the actual test size. Once the test is loaded to this area all the chains covering it are destroyed.

TAKEB(b) command may be issued in every case the area assigned to a test or other program has been released. The chains destroyed previously have to be restored starting from the address b. This action is unconditional and when it is completed the operating system may resume its normal operation.

It should be noted that TAKEB BASE command issued when the state table is cleared results in creation of the whole free memory. In the actual system implementation the use is made of this possibility.

All memory-oriented events have been given a very high priority in the operating system to make the event decoding as quick as possible. In addition, to speed up their execution the memory management routines have been coded as optimally as possible resulting in some 20-25 assembly instructions for the cell give and the cell take subroutine in case there is no need to send system overload or system continue supervisory messages.

The memory mamagements routines have been coded generally in the sense that they are valid for any state table of the structure shown in Fig. 4.

VI. Hitherto experiences

All the free memory routines have been coded as subroutines in PLAN assembly language. The whole operating system is operational now and some experiences from its behaviour in the normal operation have been gained. With any number /i.e. ≤ 21/

of TTY-7071 terminals connected to the front-end processor via a multiplexor and the host ODRA-1305 computer connected to it via an inter-processor buffer a considerable improvement in response times and throughput capabilities has been noticed by all interactive users. More quantitive analysis of those problems will be the subject of seperate studies.

In order to monitor internal states of the operating system with particular regard of the free memory behaviour some monitoring mechanisms have been built-in. In the normal system operation with 8 terminals connected overflow conditions have never been reached, except for an artificial simultaneous listing of GEORGE-3 files. The overflow conditions have been simulated by decreasing the free memory size and even then the overflow control procedures made reaching higher overflow levels almost impossible.

A general conclusion has been drawn that the overflow control mechanism is "too powerful" for the current release of the operating system, thus allowing for its future development. The real use of the mechanism should be made in an operating system capable of controlling remote terminals and a remote batch station simultaneously.

VII. FINAL REMARKS

The free memory organization as presented here-in meets all the requirements set forth at the early stage of the problem formulation, i.e. it allows for efficient transfers of data and supervisory messages, makes possible execution of tests of various and un-predetermined sizes and prevents any system break-down due to memory overflow conditions. Some mechanisms like overflow control mechanisms have been designed slightly "ahead of time" to allow for future system extentions. Any modifications of such memory parameters like cell length, chain length/s/ /the same or different for every chain/ and number of chains can be easily carried out without any impact on the memory management procedures. This ma-

kes the memory organization suitable for a broader class of communication operating systems.

REFERENCES

[1] C. Newport, J. Ryzlak, "Communication Processors", Proceedings of the IEEE, vol. 60, pp. 1321-1332, November 1972

[2] D.L. Mills, "Communication Software", Proceedings of the IEEE, vol. 60, pp. 1333-1341, November 1972

[3] E. Yourdon, "Design of On-line Systems, Prentice-Hall, Inc., Englewood Cliffs, 1972

[4] E. Bilski, L. Budzianowski, T. Muehleisen, E. Sorokin, J. Wietrzych, W. Żabnieński, "Opis emulatora komputera typu FEP dla m.c. ODRA-1325", komunikat ICT, Wrocław 1978

[5] E. Bilski et al., "Algorytmy i opis funkcjonalny systemu operacyjnego komputera komunikacyjnego, raport ICT, Wrocław 1978

[6] Data Communication and Interrogation, ICL, London 1973

Generalized bin-packing problems in processor scheduling

by A.Iványi and I.Kálmán /ELTE, Budapest/

Introduction

A number of important problems (processor scheduling, memory allocation, stock-cutting, coil slitting, cable-length optimization etc.) leads to the same mathematical model — to the so called bin-packing problem [1]. In our paper we deal with problems of processor scheduling.

We consider the following model of a computing system. There is given a set of processors $P=\{P_1,\dots P_m\}$ with speeds $\Pi=\{p_1,\dots p_m\}$. There is also given a set of tasks $T=\{T_1,\dots T_n\}$ with running times $\tau=\{\tau_1,\dots,\tau_n\}$ (on a processor with unit speed).

We call a scheduling nonpreemptive when a task cannot be interrupted once it has begun execution; that is, it must be allowed to run to completion. In general, preemptive scheduling permits a task to be interrupted and removed from the processor under the assumption that it will eventually receive all its required execution time, and there is no loss of execution time due to preemptions (i.e., preempted tasks resume

execution from the point at which they were last preempted).

In their paper Coffman, Leung and Ting [2] investigated the following problems for independent tasks, nonpreemptive scheduling and identical processors.

P1. To find the minimal number of necessary processors (m_{min}) to run the tasks for a fixed deadline (d).

P2. To find the minimal deadline (d_{min}) for a fixed number of processors (m) and given set of tasks.

P3. To find the maximal number of processable tasks (n_{max}) for a fixed deadline (d) and for a fixed number of processors (m).

We consider these problems in the following cases.

1.Preemptive scheduling of independent tasks on identical processors.

2.Preemptive scheduling of independent tasks on processors with different speeds.

We consider scheduling of jungle-structured task systems, too.

1.Preemptive scheduling of independent tasks on identical processors

We have a task system $T=\{T_1,\dots,T_n\}$ consisting of independent tasks with running times $\tau=\{\tau_1,\dots,\tau_n\}$ and m identical processors.

Let us consider problem P2. In Coffman's book the following algorithm is proposed.

Algorithm A [1, Algorithm 2.4].

1. Let $X=\sum_{i=1}^{n}\tau_i$ and $\tau_{max}=\max_{1\le i\le n}\tau_i$, the largest running time, and $d=\max\left(X/m,\tau_{max}\right)$.

2. Initially none of the processors has been assigned to any tasks. Do step 3 for $T_1,\dots T_n$.

3. Let tasks $T_1,\dots T_{i-1}$ already be assigned to processors $P_1,\dots,P_{j-1}$ in the interval $(0,d)$ and processor P_j in the interval $(0,t)$ for some $t<d$. Consider task T_i. If $t+\tau_i \le d$, assign task T_i to P_j in the interval $(t,t+\tau_i)$. Otherwise assign T_i to P_j during (t,d), and to P_{j+1} during $\left(0,\tau_i-(d-t)\right)$.

In Coffman's book it is shown that for Algorithm A holds the following

Theorem 1 [1, Theorem 2.5].

Algorithm A determines a minimal-deadline preemptive schedule for systems of independent tasks on m identical processors.

/Then this minimal deadline is $d=\max\left(\frac{X}{m},\tau_{max}\right)$./

For problem P1 and P3 we proved the following assertions.

a/ Consider problem P1.

Assertion 1.

Let $0<\tau_i\le d \quad (i=1,\dots,n)$. Then the minimal number of necessary processors equals the upper integral part of the quotient of the sum of τ_i-s and the deadline d, that is

$$m_{min}=\left\lceil\frac{\sum_{i=1}^{n}\tau_i}{d}\right\rceil.$$

Proof

On one processor we can execute k tasks, for which $\sum_{i=1}^{k} \tau_i \leq d$. If we have to execute n tasks, then for the number of necessary processors j holds the following inequality:

$$(j-1)d < \sum_{i=1}^{n} \tau_i \leq jd.$$

Divide each side of the unequality by d then we obtain that

$$j-1 < \frac{\sum_{i=1}^{n} \tau_i}{d} \leq j.$$

That is

$$j = \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{d} \right\rceil.$$

Assertion 2

Step 2 and step 3 of Algorithm A determine a schedule which requires a minimal number of identical processors.

Proof

If there is a task to be assigned to two processors (eg. P_j at the end of the interval $(0,d)$ and P_{j+1} at the start of the interval $(0,d)$), then the two part of the interval $(0,d)$ are disjunct, because for each task of the system holds $\tau_i \leq d$. So this schedule is a valid preemptive schedule.
The algorithm (step 2 and step 3 of Algorithm A) schedules the tasks immediately one after the other, so there is no idle time in the schedule constructed by the algorithm, at most on the last processor which is used. Then the number of required processors is minimal.

b/ Let us consider problem P3.

Let the task system T, the deadline d and m identical processors be given. Let $0<\tau_i \leq d \ (i=1,\dots,n)$. Then $m \leq n_{max} \leq n$. If we know the largest running time we can give a little better estimation, namely:

Assertion 3

Let $0<\tau_i \leq \alpha < d \ (i=1,\dots,n)$, then

$$\min\left\{n, \left[\frac{m \cdot d}{\alpha}\right]\right\} \leq n_{max} \leq n.$$

Proof

It is clear that $n \geq n_{max}$. Suppose that the tasks are in non-decreasing order of their running time and that we can execute the first k tasks on the m processors but we cannot execute the first k+1 tasks. Then we cannot execute more tasks on the m processors with any other ordering of tasks, than k. For this k

$$\sum_{i=1}^{k} \tau_i \leq m \cdot d < \sum_{i=1}^{k+1} \tau_i .$$

Since $\tau_i \leq \alpha \ (i=1,\dots,n)$, $\sum_{i=1}^{k+1} \tau_i \leq (k+1)\alpha$. From these unequalities we obtain $md<(k+1)\alpha$. Divide each side of this unequality by α, then

$$\frac{m \cdot d}{\alpha} < k+1,$$

that is

$$\left[\frac{m \cdot d}{\alpha}\right] \leq k.$$

To execute the maximal number of tasks we propose the following algorithm.

Algorithm B

Let $\tau_1 \leq \tau_2 \leq \ldots \leq \tau_n$. Perform Algorithm A with the following modification: if there is no more free processor then the algorithm finishes its work.

For Algorithm B holds the following

Assertion 4

Algorithm B determines a schedule with n_{max}.
(That is if we schedule the tasks in nondecreasing order of their running time then the number of processable tasks on m identical processors will be maximal for a given deadline d.)

Proof

Suppose that we can execute k tasks by Algorithm B. Because the tasks are in nondecreasing order of their running time each task not scheduled has larger running time than any scheduled task. Thus an interchange between a scheduled and a not scheduled task cannot decrease the time required by the execution of k tasks. Then this interchange cannot cause the increase of value of k. An interchange between the scheduled tasks, of course, cannot change the value of k, too. Soo k is maximal.

2. Preemptive scheduling of independent tasks on processors with different speeds

Let $T = \{T_1, \ldots, T_n\}$ be a task system consisting of independent tasks with running times $\tau = \{\tau_1, \cdots, \tau_n\}$ on a processor with unit speed . We have m processors $P_1, \ldots P_m$ with speeds $p_1, \ldots p_m$. Let $X_i = \sum_{i=1}^{i} \tau_i$ and $Q_j = \sum_{i=1}^{j} p_i$.

To determine a preemptive shedule with minimal deadline in Coffman's book there is proposed an algorithm. The description of this algorithm is long, therefore we omit it.

This algorithm constructs a schedule with deadline d, where

$$d = \max\left[\max_{1\le j\le m}\frac{X_j}{Q_j}, \frac{X_n}{Q_m}\right].$$

Theorem 2 [1, Theorem 2.11].

The proposed algorithm constructs a preemptive schedule with minimal deadline for T on $P_1,\ldots,P_m$.

About m_{min} and n_{max} we can prove the following estimations.

a/

Assertion 5

We have the task system T and the deadline d.

Let $p_{min} = \min_{1\le i\le m} p_i$. Then for the number of necessary processors m_{min} holds

$$m_{min} \le \left\lceil \frac{\sum_{i=1}^{n}\tau_i}{d\cdot p_{min}} \right\rceil.$$

Proof

For the number $j=m_{min}$ of necessary processors (to run the n tasks) holds the following:

$$d\sum_{k=1}^{j} p_k \ge \sum_{i=1}^{n}\tau_i > d\sum_{k=1}^{j-1} p_k.$$

Since $p_i \ge p_{min}$ $(i=1,\ldots,n)$ therefore

$$d\sum_{k=1}^{j-1} p_k \ge d(j-1)p_{min},$$

so

$$\sum_{i=1}^{n}\tau_i > d(j-1)\cdot p_{min}.$$

Divide each side of the unequality by dp_{min}, then we have

$$\frac{\sum_{i=1}^{n} \tau_i}{d \cdot p_{min}} > j-1,$$

that is $j \leq \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{d \cdot p_{min}} \right\rceil$, which means $m_{min} \leq \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{d \cdot p_{min}} \right\rceil$.

b/ Let us consider problem P3.

Assertion 6

Let $\tau_1 \leq \tau_2 \leq \ldots \tau_n$ and $p_1 \leq p_2 \leq \ldots \leq p_m$.

Let $0 < \tau_i \leq \alpha \leq dp_m$. Then for the maximal number of processable tasks n_{max}

$$\min \left\{ n, \left\lceil \frac{d \cdot \sum_{j=1}^{m} p_j}{\alpha} \right\rceil \right\} \leq n_{max} \leq n \text{ holds}$$

(if there is a valid schedule for these n_{max} tasks on the m processors).

Proof Let $k = n_{max}$.

It is clear that $n \geq k$. Suppose that we can execute the first k tasks on the m processors but we cannot execute the first $(k+1)$ tasks. Then we cannot execute more tasks on the m processors with any other order of tasks than k because the tasks are in nondecreasing order. For this k holds that

$$\sum_{i=1}^{k} \tau_i \leq d \sum_{j=1}^{m} p_j < \sum_{i=1}^{k+1} \tau_i .$$

Since $\tau_i \leq \alpha$ $(i=1,\ldots,n)$, then $\sum_{i=1}^{k+1} \tau_i \leq (k+1)\alpha$. From these unequalities we obtain

$$d \sum_{j=1}^{m} p_j < (k+1)\alpha .$$

Divide each side of the unequality by α, then we obtain

$$\frac{d\sum_{j=1}^{m} p_j}{\alpha} < k+1,$$

that is $\left\lceil \frac{d\sum_{j=1}^{m} p_j}{\alpha} \right\rceil \leq k$, which means that $n_{max} \geq \min\left\{n, \frac{d\sum_{j=1}^{m} p_j}{\alpha}\right\}$.

The following algorithm gives the accurate value of n_{max} for given T, d and m.

Algorithm C

1. Let i=1.
2. Let n_{max}=i.
3. Let $d_{min}=\max\left[\max_{1\leq j\leq m} \frac{X_j}{Q_j}, \frac{X_i}{Q_m}\right]$.
4. If d_{min} d then let i=i+1 else go to 6.
5. If i>n then stop else go to 2
6. $n_{max}:=n_{max}$-1. Stop.

3. Scheduling of jungle-structured task systems

In Coffman's book the following definitions are used.

Definition 1

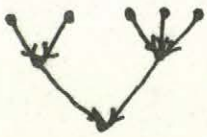
A directed, acyclic graph forms a forest, if either each vertex has at most one predecessor, or each vertex has at most one successor.

Definition 2

If a forest has in the first case exactly one vertex with no processor, or in the second case, exactly one vertex with no successors, it is also called a tree.

Examples

Tree in the first case

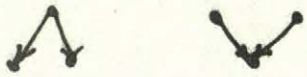


Tree in the second case

Forest; at most one predecessor

Forest: at most one successor

No forest:



According to these examples Coffman's definition is equivalent to the following one: a forest is a union of the trees of the same type.

We propose a new definition.

Definition 3

The union of trees of any type is a jungle.

Of course, in this case the forest is a special case of jungle. Using this definition, we can a bit extend many theorems about forest-structured task systems.

Let us consider an example.

In Coffman's book we can find the following definition.

Let the level of a node x in a directed, acyclic graph be the maximal number of nodes including x on any path from x to a terminal task./In a jungle, there is exactly one such path./ A terminal task is at level 1.

Definition 4 [1,p.54]

Whenever a processor becomes available, assign it to an unexecuted ready task at the highest level (forthest from a terminal). A schedule thus constructed is called a level schedule.

(Let a task be ready when all its predecessors have been executed.)

Theorem 3 [1, Corollary 2.1]

Let $T=\{T_1,\ldots T_n\}$ be a forest-structured task system with $\tau_i=1$ $(i=1,\ldots,n)$. Then the level strategy constructs a minimal-deadline nonpreemptive schedule for T on m identical processors.

Proof.

Let $\prec$ denote the partial order of tasks corresponding to the given forest.

If the level strategy does not generate a minimal - - deadline schedule for the task system $(T,\prec)$ the strategy cannot generate a minimal-deadline schedule for the tree- - structured task system $(T',\prec')$ formed by adding a new task that is an immediate successor of exactly the terminals of $(T,\prec)$.

The result follows from Theorem 2.1. in Coffman's book applied to $(T',\prec')$.

We can prove the following assertion on the same way.

Theorem 4

Let $T=\{T_1,\ldots T_n\}$ be a jungle-structured task system with $\tau_i=1$ $(i=1,\ldots,n)$. Then the level strategy constructs a minimal deadline nonpreemptive schedule for T on m identical processors.

References

[1] E.G.Coffman, Computer and job-shop scheduling theory, Wiley and sons, New York, 1975, pp.208-227.

[2] E.G.Coffman, J,Y.-T Leung, D.Ting, Bin packing problems in storage and processor allocation, in. K.M.Chandy, M.Reiser (Ed.), Computer performance, North Holland, Amsterdam, New York, 1977, pp.327-339.

Kovács Győző - Németi Tibor:

Evaluation of a Time-sharing Computer System

1. Introduction

Computer centers, in order to increase their efficiency, attempt to utilize the computer to the greatest degreee possible, attempt to optimize the workload of the computer system, e.g.

- to execute the greatest possible number of tasks in the shortest possible time while providing the greatest possible degree of user comfort;
- to achieve maximum load on all the resources /CPU, peripherals, etc./ of the system;
- to achieve maximum utilization of the central processor while the peripherals are not so loaded;
- to achieve quick turn-around for test runs;
- to ensure that big task are done by the prescribed deadlines.

The computer center chooses those optimization criteria which help the center achieve its goals to the greatest extent. In an earlier paper measurements and investigations concerning a specific computer were published in addition to an example demonstrating the evaluation of the maximum loading of time-sharing computer system using bench-mark programs.

The current work gives a general solution for the problem, in other words how is it possible by using knowledge of the environmental parameters /hardware configuration, operating system, applications/, to optimaly shape the work of the time-sharing system, that is to utilize the computer to the limits of its capabilities.

2. Measurement of the efficiency of the time-sharing system

The capacity of a computer /CPU/ is characterized in various ways, for instance by the number of operations execute per second. The user of the computer system is only partially interested in the performance of the CPU; he is more intereste in the performance of the total system /hardware, software, environment/ since only the optimal operation of the total system means a tangible advantage for him.

In a time-sharing environment during the normal operation of the system jobs arrive from the terminal /conversational mode/ or from the card-reader /batch mode/ randomly in time, so it is difficult or even impossible to measure the performance determined by the number of operations per second. It is therefore necessary to find a parameter or parameters which can be easily observed during operation and which characterize the load state of the system. The system administrator can then use this parameter to increase or decrease the load, or to prevent the overloading of the system /the execution of

the tasks is slowed, response times increase rapidly, in other words the performance of the system declines/.

In the investigated time-sharing system the programs and the data are in the virtual memory of the machine. By increasing the number of programs running simultaneously the performance of the machine /number of instructions executed in a time unit/ increases together with an increase of the paging rate also. At a certain number of concurrently executing programs the system reaches its performance maximum; starting further concurrent programs results in the decline of useful performanc the system slows down, becomes overloaded.

The above situation is easily achieved in an "artificial environment", for example by running a program several times in parallel, which satisfies a few simple conditions /pages during its execution, can be run several times in parallel, reports the executions of a given number of instructions and number of page faults that is the measure of the work performec - together with the execution time/.

The measurement program utilized contains an arbitrary number of instuctions /n/, which when executed alone executes in time T_1. If several of these are run in parallel with each other, then the execution times of the individual programs /T_2, T_3, ... , T_i/ increase, that is

$$T_2, T_3, \ldots, T_i > T_1$$

The number of instructions executed per time unit - by our definition the performance of the system - in case of several programs running in parallel is

$$P = n \sum_{i=1}^{k} \frac{1}{T_i}$$

where k is the number of programs running concurrently.

Since n is a constant the examination of the value of $\frac{P}{n}$ is sufficient to determine the state of the system. Figure 1., curve A shows the results of the measurement using the above program, that is the change in the performance as function of the number of concurrently executing programs /k/. It can be seen that with the given measurement program up to the execution of six concurrent programs / $\geq$ 6/ the machine executes an increasing number of instructions in a time unit /gives increasing performance/, for k=6 the performance is at a maximum. If seven or more programs are run concurrently /k $\geq$ 7/, the administration of the system increases; the paging rate cannot increase adequately with the number of programs, the performance declines.

In the system - using special programs- several operational parameters can be measured during the operation of the system, namely

a/ utilization of the central processor /in percent/,

b/ paging rate,

c/ characteristic value of the trafic between main and secondary memory,

d/ CPU response time,

e/ number of task running concurrently in batch and conversational mode,

f/ number of jobs waiting for processing /spool-in/, etc.

In figure 1. the change in parameters a/ and b/ are also shown as a function of the number of concurrent tasks /curves B and C/. The measurements showed that the load of the CPU varied proportionally with the performance until the overloaded state is reached and that no connection can be demonstrated with the achivement of the overload point. Overloading occured in the confirgurations used in the measurement at a certain page rate /C = 38/, this can also be shown for other programs.

With different types and sizes of programs the value of k characterizing overloading can be different. The program described used 25o kbytes of memory, which in the case of k=6 means the use of 1.5 Mbytes of virtual memory. Since at the time of the measurement the size of pageable core was about 36o kbytes, the ratio between the two is about 4:1. Using a larger program /e.g. 75o kbytes/ which was organized in similar manner for paging, the maximum system performance, that is the overload point was reached at k=2,

at the same characteristic value of paging /C $\leq$ 38/ as before.

Figure 2 shows the same measurement, but with a different configuration. The difference is that now the secondary memory area used for paging is accessible through one channel as opposed to the two available during the previous measurement. It can be seen that the maximum performance is greatly decreased, the system gives the performance maximum for k=4 concurrent programs, above this number the system is overloaded.

The operation of the system has to be organized in such a way as to avoid overload. To control system load the system administrator has various facilities, for instance:

- with an operator command the maximum number of batch and/or interactive tasks can be determined;
- jobs with long run times that are waiting for execution can be prevented from initiation;
- execution of individual programs can be pended;
- the relative priority of batch and interactive jobs can be changed;
- the operation of the entire system may be suspended if the available on-line disk area becomes saturated, etc.

3. Possibilities of increasing the maximum performance of a time-sharing system

Without changing the basic characteristics of the system

- hardware configuration,
- operating system,
- application programs,

there are still several possibilities for increasing the performance of the system, at the same time postponing the overloaded state. The test programs /Figure 3/ which were used for the performance measurements, were characteristic of the daily operation of the system /large data movement tape--handling, average CPU usage, conversational/.

The essential feature of the measurement was that the programs ran in a given order - some programs several times cyclically - while the operational conditions changed from measurement to measurement.

The results of the measurements can be evaluated with the following measured data:

- total run time /sum of the run times of the individual programs/ R_T
- turn around time, time elapsed from the initiation of the first program until the completion of the last program - T_t
- the ammount of time used for the execution of a given group of programs, P

- central processor time, T_c, the time during which the complete task sequence utilized the CPU of the machine,
- efficiency indicators for the totalled run times and central processor times

$$E_R = \frac{T_c}{R_T}$$

$$E_t = \frac{T_c}{T_t}$$

By appropriate system generation, varying the operational limits /e.g. the size of on-line memory/, administrative actions, etc. several measurements and comparisons were possible; of these a few characteristic results will be described in the sequel. Especially those will be selected which can be used to increase the performance of the computer system.

3.1. Placement and accessibility of the virtual memory

The placement of the virtual memory /2-5 disks/ and the mode of its access /one or two channels/ can be changed by software means during system generation. The effect of varying the channels can be seen from Figures 1. and 2. The time required for the compilation of a FORTRAN program /P/ changes in the case of varying the number of channels as follows:

Concurrent programs channel	1	2	3	6	Remarks
P_1 /sec./	25	38	75	1o5	ane channels
P_2 /sec./	14	25	5o	7o	two channels
P_1/P_2	1,79	1,52	1,5	1,5	

thus it can be seen that varying the number of channels produced a 5o % larger system performance.

Varying the number of disks used for the placement of the virtual memory also resulted in increased performance.

Mode	R_T sec.	T_t sec.	T_c sec.	E_R %	E_t %
2 disk	5982o	1182o	1671	2,79	14,14
5 disk	2550o	5o4o	139o	5,45	27,57
Ratio 2/5	2,35	2,35	1,2	X	X

The table shows that the 5 disk system gives better performance when compared to the two disk system due to the greater dispersal of the virtual memory.

3.2. Saturation of the on-line memory area

User experience showed that when the system was saturated, that is when more than 8o % of the on-line memory area /6oo Mbytes/ was in use, the system slowed; at saturation stopped completely. Thex following measurement corroborates experience.

Loading state	$R_{T_{sec.}}$	$T_{t_{sec.}}$	$T_{c_{sec.}}$	$E_{R_{\%}}$	$E_{t_{\%}}$
~ 85 % - 5oo MB	339oo	642o	1482	4,37	23,1
~ 4o % - 25o MB	2544o	5o4o	139o	5,46	27,6
Ratio ~ 85/~ 4o	1,33	1,27	1,o7	X	X

System performance declines as saturation approaches, the change is about 3o %, this can be alleviated by operator intervention /freeing of memory areas, rolling back of data to tape/.

3.3. Effect of the priorities of batch and interactive programs

In the time-sharing system priorities can be used to give advantages to batch or time-sharing programs. In a function of the daily program selection significant performance increases can be achieved by giving appropriate priorities; this is shown by the following two measurements.

Priority	R_T sec.	T_t sec.	T_c sec.	E_R %	E_t %
Batch	1992o	438o	148o	7,43	33,8
Interactive	2622o	618o	2242	8,55	36,3
I/B ratio	1,32	1,41	1,51		

Priority of batch programs increased efficiency, since the machine could perform more work in a time unit, it did not have to wait for the interactive programs. A consequence of this - and this is a drawback - is that the level of terminal service declined. This kind of efficiency increasing measure may only be applied when there are many batch programs in the system and few users working on terminals.

3.4. Measurement of overloading

By varying the concurrent running of the meas ing programs the overload effect discussed in section two can be demonstrated The results of the first measurement show that running the programs sequentially results in the underutilization of the system; the system can be further loaded. By running some of the programs in parallel /page rate C 3o/ maximum system performance can be achieved. When all programs run together the system becomes overloaded. The efficiency indicator

related to throughtput /turn around time/ E_t, is greatest near the maximum; this is when the system gives the greatest performance in a time unit. The results of the measurements are as follows.

Run mode	$R_{T_{sec.}}$	$T_{t_{sec.}}$	$T_{c_{sec.}}$	$E_{R_{\%}}$	$E_{t_{\%}}$	m
Sequentiul	426o	426o	1589	37,3	37,3	1,22
Mixed /optimum/	1764o	6ooo	2739	15,5	45,65	2,28
Parallel	2544o	5o4o	139o	5,46	27,6	3,86

where $m = \frac{\text{Virtual memory utilized}}{\text{Pagable core}}$

4. Increased efficiency by varying the hardware configuration

It is evident that with a larger system the performance can be increased. Replacement of certain elements of the system or the increase of the number of devices brings the wanted results, only if the factors degrading system performance are discovered, and so performance can be increased with a relatively small investment.

The achieve this discovery it is necessary to perform measurements, bench-mark tests, observations of the daily operation of the system similar to the ones described. A

problem can be solved in several ways, obviously one must choose the most efficient and from a financial side advantageous solution. An example:

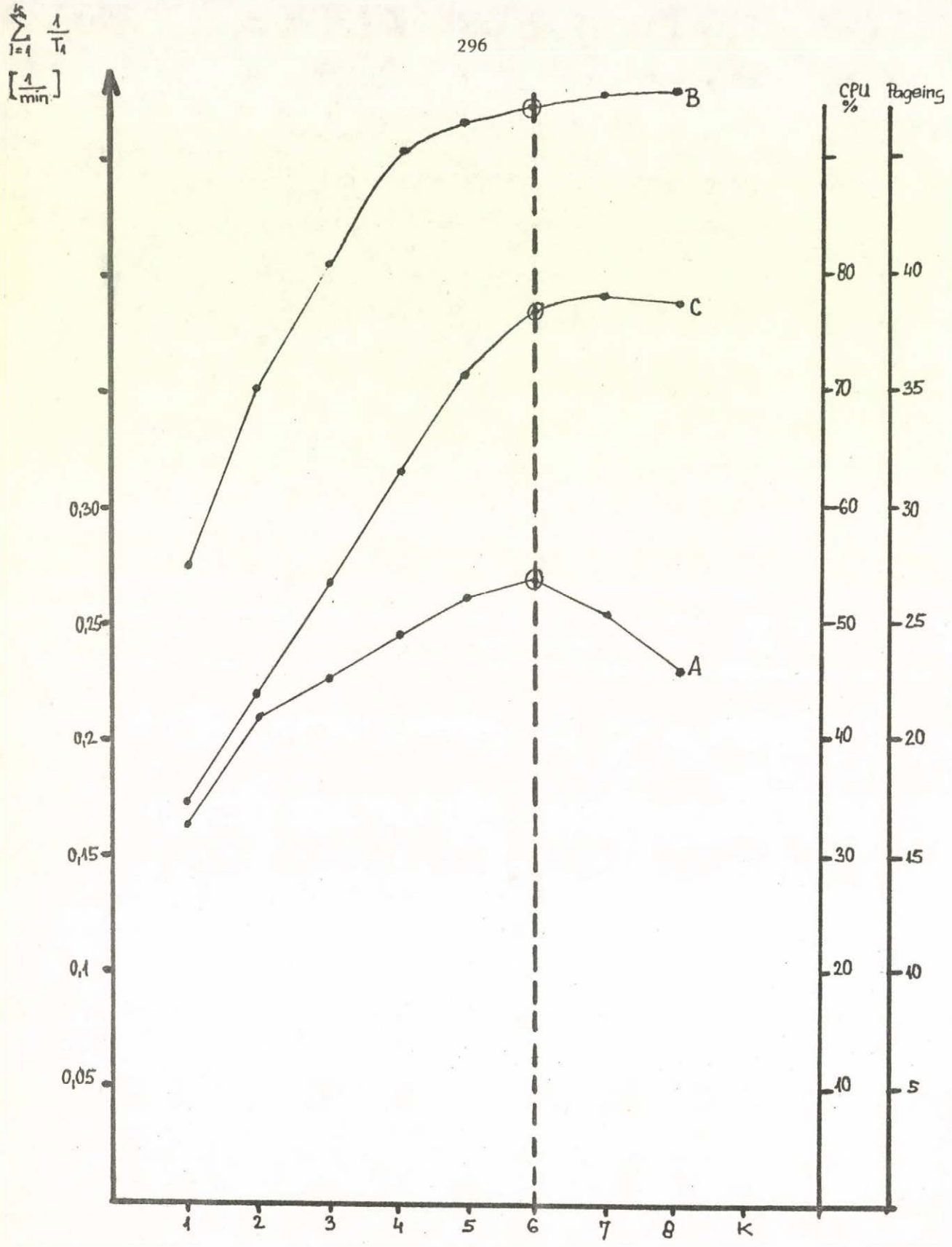
The system becomes overloaded quickly, yet CPU utilization is low /4o-5o %/. The cause: the paging scheme functions incorrectly /e.g. Figure 2./. Several solutions are possible, for instance adding to the system

- another channel, perhaps disk controller, or
- new disk units, or
- larger central memory.

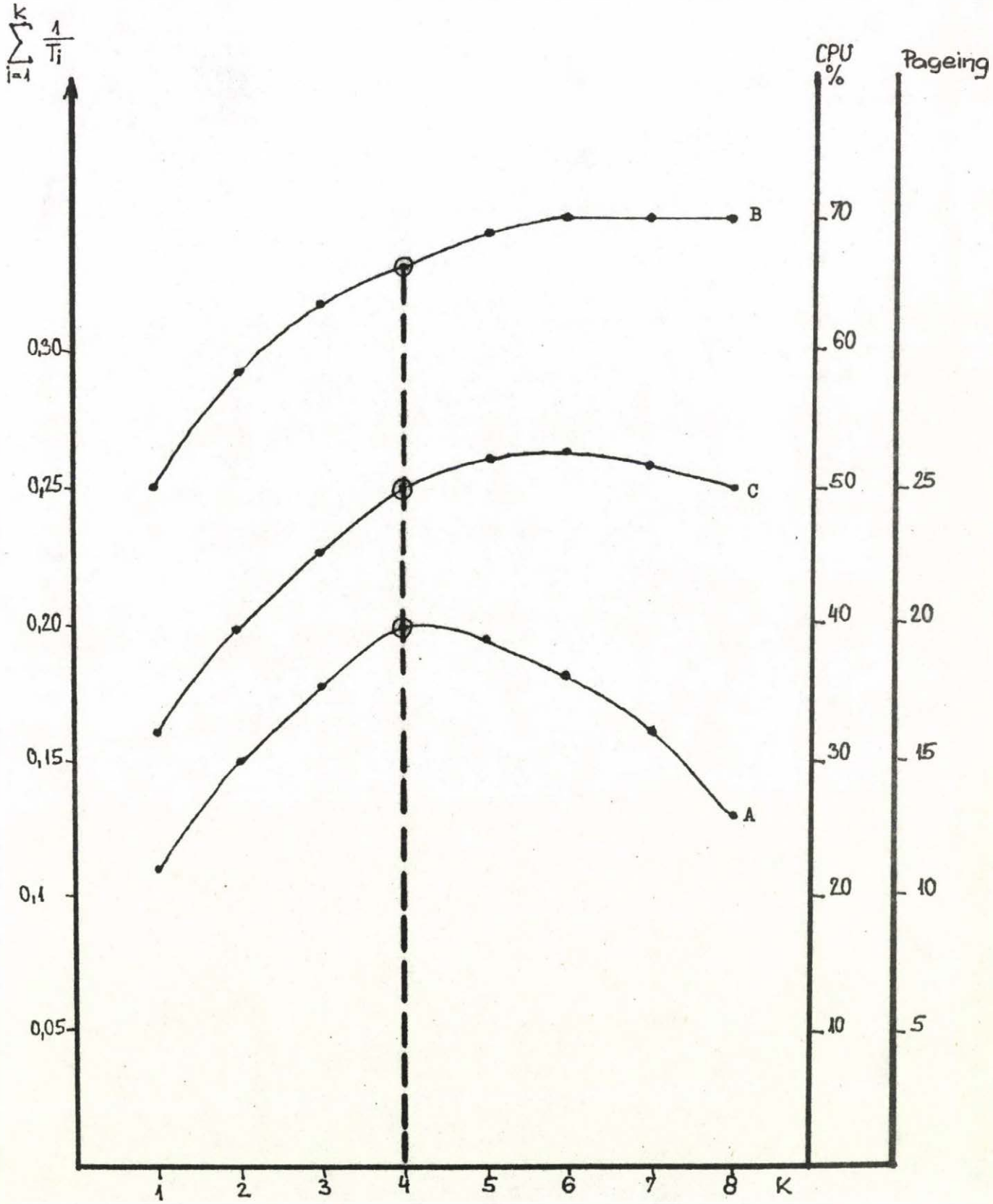
Summarizing, it can be stated that by a continuous observation and measurement of the time-sharing system even without changing the basic system elements /hardware and software/ significant performance increases can be achieved by simple software or administrative intervention.

- . -

We would like to acknoledge the help of - Szőllősi Etelka and Frank Lajos in writting the measurement programs and in performing the measurements.


$\sum_{i=1}^{k} \frac{1}{T_i}$
$\left[\frac{1}{min}\right]$
CPU %
Paging
B
C
A
0,30
0,25
0,2
0,15
0,1
0,05
80
70
60
50
40
30
20
10
40
35
30
25
20
15
40
5
1
2
3
4
5
6
7
8
K

1 Figure


$\sum_{i=1}^{K} \frac{1}{T_i}$
CPU %
Pageing
0,30
0,25
0,20
0,15
0,1
0,05
70
60
50
40
30
20
10
25
20
15
10
5
B
C
A
1
2
3
4
5
6
7
8
K

2 Figure

3. Fig.

N a m e	Size Page /4 Kbyte/	Typ
TAPE-handling	35	Dialóg
SORT	43	Dialóg
COBOL + LINKAGE	64+26	Dialóg
FORTRAN Compilation	29	Dialóg
PL 1	33	
LINKAGE	56	
Appl-phase	56	
CPG	11o	Batch
ASSEMBLER	51	
LINKAGE	1o3	

Bibliography

1.Calingaert, P.: System performance evaluation: survey and appraisal. Com ACM vol 1o no 1 pp. 12-18 /jan. 1967/.

2.Ferrari, D.: Workload characterisation and selection in computer performance measurement, Computer, vol 5 no. 4. pp 18-24 /july/aug. 1972./

3.Karush, A.D.: Two approaches for measuring the performance of time sharing systems. Proc.2nd ACM Symp. Op Sys 1969. 159-166 pp.

4.Kolence, K.W.: A software view of measurement tools. /Datamation vol 17 no 1. pp 32-38 /jan. 1. 1971./

5.ACM Proceedings of workshop on system performance evaluation Harward Univ. Cambridge Mass /April 1971/

5.Anderson, M.A. and Sargent, R.G.: Bibliography 31: Modelling, evaluation and performance measurements of time sharing computer systems.

6.Arbuckle, R.A.: Computer analysis and throughput evaluation. Computers and Automation vol 15. no 1. pp 12-15., 19. /Jan. 1966./

7.Batson, A. and Brundoge, R.: Performance measurements on a virtual memory computer system in a batch processing environment. Proc. ACM Workshop 1971 pp 214-226.

8.Scheer, A.L. An analysis of time shared computer system. MIT Press, Cambridge Mass /1967/

9. Smith, J.M.: A review and comparison of certain methods of computer performance evaluation. Computer Bulletin vol 12 no 1. pp 13-18. /may. 1968./

lo. Stimler, S.: Some criteria for time-sharing system performance. Comm ACM vol 12. no 1. pp 47-63 /jan. 1969/.

11. Wood, D.C. and Forman, E.H.: Throughput measurement using a synthetic job stream. Proc. AFIPS 1971. FJCC pp 51-56.

12. Blatny, J. et al.: On the optimisation of performance of time-sharing systems by Simulation. C/ACM Vol 15. no 16. pp 411-42o /June 1972./

13. Wikles, M.V.: Automatic load adjustment in time-sharing systems. Proc ACM-SIGOPS workshop on System performance pp 3o8-32o /April 1971./

14. Kovács Gy. - Németi T.: Methods and Tools used in the Institute for Coordination of Computer Techniques for Optimising the Time-sharing system. COMNET Conference IFIP, 1977.

THE EFFECT OF PAGE SIZE ON THE SPEED

A. IVÁNYI AND Z. PÓKOS

Eötvös University of Budapest

In this paper, we generalized a result of Fagin and Easton [1]. They proved, that miss ratio and page size are essentially independent in the independent reference model of program behaviour [2].

First let us see their main result. Let MISS be the expected working set miss ratio in the independent reference model with expected working set size CAP pages. Now form blocks, by combining the B pages with the next highest probabilities of reference into a second block, and so on. Let $MISS^*$ be the expected working set miss ratio when all data are moved in blocks, and when the expected working set size is again CAP pages, that is CAP/B=C blocks. Easton and Fagin proved, that

$$\left| MISS - MISS^* \right| < \frac{2}{C} + \frac{33}{C^2} \quad .$$

We proved, that this unequality remains true, not only in the independent reference model, but in a more general Markov reference model too. Thus, if the expected working set size /in blocks/ is sufficiently large, then the miss ratios in the blocked and unblocked cases are approximately equal. This result gives a theoretical justification to the empirical observation, that in some computing systems with paged, two level storage

hierarchy, long term miss ratio is roughly independent of page size. The research was sparked by the empirical observation of Bennett [1], who examined a page reference trace from the IBM Advanced Administrativ System /A.A.S./, a large internal IBM data management system. Bennett found no consistent relationship between miss ratio and page size; for some main /first level/ memory sizes the miss ratio was slightly larger for the larger page size, and for other main memory sizes slightly smaller. In all cases the size of main memory had a vastly greater effect on miss ratio then did page size, if the page size was sufficiently large /at least 1500 bytes/.

After this introduction let us come to the point. First we describe the theorem of Easton and Fagin formally. Let $p_1, p_2, \ldots, p_n$ be a probability distribution /that is $\sum p_i = 1$, and each $p_i \geq 0$ /. Assume that at time t page i is referenced with probability p_i, independent of past history. The expected working set miss ratio /with window size T/ is the probability that the page referenced at time t was not one of the pages referenced over the course of the previous T /not necassarily distinct/ references. Under the independent, time invariant assumption of independent reference model, it is clear that the expected working set miss ratio is independent of t for $t > T$. Let CAP be the expected working set size, that is the expected number of distinct pages to appear over the course of T references. Define MISS/CAP/ to be the corresponding expected working set miss ratio. Thus MISS/CAP/ is the expected working set miss ratio with window size T, where the expected working set size

with window size T is CAP pages.

We will now describe the blocked case. Let B /the block size/ be a positiv integer /dividing n as we assume for convenience/. Assume, that $p_1 \geqslant p_2 \geqslant \ldots \; p_n \geqslant 0$ and let

$$u_i = \sum_{j=1}^{B} p_{(i-1)\cdot B+j} \qquad 1 \leqslant i \leqslant n/B$$

Thus $u_1 = p_1 + \ldots + p_B$, $u_2 = p_{B+1} + \ldots + p_{2B}$, and so on. This corresponds to combining the B pages with the highest probabilities of reference into a block, the B pages with next highest probabilities of reference into a second block, and so on. The blocked case corresponds to the independent reference model with block probabilities $u_1, u_2, \ldots, u_{n/B}$. Define $MISS^*$/CAP/ to be the expected working set miss ratio /over the probabilitty distribution $u_1, u_2, \ldots, u_{n/B}$ when all data are moved in blocks, and when the window size is chosen so, that the expected working set size is CAP/B blocks. /CAP/B blocks contain the same number of bytes as CAP pages, and this is the quantity we hold fixed in comparing the blocked and unblocked cases./ Let C=CAP/B, and write MISS and $MISS^*$ for MISS/CAP/ and $MISS^*$/CAP/. The main result of Easton and Fagin:

$$\left| MISS - MISS^* \right| < \frac{2}{C} + \frac{33}{C^2} \; . \qquad /1/$$

Note that statement /1/ is a distribution free result, that is the error terms do not depend on the values of the of the p_i, or even on n, the number of pages.

Now we give an expression for MISS/CAP/, the expected working set miss ratio when the expected working set size is CAP pages. Te expected working set size, that is the expected number of distinct pages which will be referenced over the course of T references, is

$$S/T = \sum_{i=1}^{n} /1/1-p_i/^T/,$$

beacause the probability that page i is referenced is $1-/1-p_i/^T$. The expected working set miss ratio with window size T is

$$M/T/= \sum_{i=1}^{n} p_i/1-p_i/^T$$

because $p_i/1-p_i/^T$ is the probability, that page i is next referenced, and page i did not appear in the last T references. Thus if S/T/=CAP, than MISS/CAP/ is by definition

$$M/T/ = M/S^{-1}/CAP//.$$

Note, that $M/S^{-1}/x//$ is well defined for each real number x between 0 and n, even if the intermediate parameter $T=S^{-1}/x/$ is not an integer, beacause S is a continuous, monoton increasing function in T. By this procedure which amounts to an interpolation, we can define $MISS/x/=M/S^{-1}/x//$ for each x with $0 \leqslant x \leqslant n$. Similarly in the blocked case, we define $MISS^*/x/= =M^*/S^{*-1}/x/$ for $0 \leqslant x \leqslant n$, where

$$S^*/T/ = \sum_{i=1}^{u/B} /1-/1-u_i/^T/ \qquad 0 \leqslant T$$

$$M^{*}/T/ = \sum_{i=1}^{n/B} u_i/1-u_i/^{T} \qquad 0 \leq T$$

Thus, if the expected working set size is CAP pages, /that is CAP/B block/ then $MISS^{*}/CAP/$ is the expected working set miss ratio in the blocked case.

Our purpose is to show, that the main result of Fagin and Easton remains true in a more realistic model. So we don't have to know more about the proof of the theorem, than that Easton and Fagin proved it analitically by using the above expressions for S/T/, M/T/, $S^{*}/T/$, and $M^{*}/T/$. Thus all we have to show is that the working set miss ratio equals M/T/ and the expected working set size equals S/T/ in the Markov model we use, too.

First describe the Markov reference model. Let's consider a homogeneous Markov chain with a transition matrix, $P=[p_{i,j}]$ and an initial distribution $\pi = [\pi_1, \ldots, \pi_n]$. Assume, that

$$P/r_{t+1}=k \mid r_t=j/ = p_{j,k} \quad \text{for each } t>0, \quad \text{and} \quad 1 \leq j,k \leq n$$

that is, the probability, that page k is referenced at time t+1 in condition that page j was referenced at time t, is $p_{j,k}$. Naturally $P/r_1=k/ = \pi_k$. We can see, that in this model the probability, that page k is referenced at time t is not independent of past history, it depends on the previous reference model was. On the basis of the real process we model /paging/, we can assume, that there exists such an integer, $s>0$ and a page r $1 \leq r \leq n$, that $p_{j,r}^{/s/} > 0$ for each j between 1 and n, where we marked the probability of s-tep transition

from j to k with $p_{j,k}^{/s/}$. This assumption means that there exists a page r, which is referenced with a positiv probability after some reference, and this is natural. But if it is true, then on the basis of Markov's theorem the Markov chain is ergodic, that is the

$$\lim_{s \to \infty} p_{j,k}^{/s/} = P_k$$

limit values exist for each j and k, and are independent of j; the numbers $p_1,\ldots,p_n$ are the unique common solution of the following equation system:

$$P_k = \sum_{j=1}^{k} P_j \, p_{j,k} \qquad 1 \leq k \leq n$$

$$\sum_{k=1}^{n} P_k = 1$$

After it, if we assume, that $\pi_k = P_k$ for each k=1,...,n, then $P/r_t = k/ = P_k$ for each k and t.

We vill prove the following

Theorem.

Let's assume, that

$$p_{j,j}^{/s/} = P_j \quad \text{for each} \quad j=1,\ldots,n, \quad s=1,\ldots,T$$

where T is the window size. Then the expected working set miss ratio in the Markov model is

$$N/T/ = \sum_{i=1}^{n} P_i/1-P_i/^T$$

and the expected working set size is

$$Z/T/ = \sum_{i=1}^{n} /1-/1-P_i/^T/$$

It can be seemed, that we want to prove, that N/T/=M/T/ and Z/T/=S/T/.

Proof:

The probability, that page j is next referenced page, and page j is not in the main memory, that is page j did not appear in the course of last T references, is the following sum:

$$B_j(T) = \sum_{\substack{k_1=1 \\ k_1 \neq j}}^{n} \sum_{\substack{k_2=1 \\ k_2 \neq j}}^{n} \cdots \sum_{\substack{k_T=1 \\ k_T \neq j}}^{n} P_{k_1} \cdot P_{k_1,k_2} \cdot P_{k_2,k_3} \cdots P_{k_{T-1},k_T} P_{K_T,j} .$$

Hence the expected working set miss ratio is

$$N/T/ = \sum_{j=1}^{n} B_j/T/$$

so we need not deal with anything else, but $B_j/T/$. Let $B_j/0/$, which is not defined till now, be P_j. Then:

$$B_j(T) = \sum_{\substack{k_1=1 \\ k_1 \neq 1}}^{n} \cdots \sum_{\substack{k_T=1 \\ k_T \neq 1}}^{n} P_{k_1} P_{k_1,k_2} P_{k_2,k_3} \cdots P_{k_{T-1},k_T} \cdot P_{k_T,j} =$$

$$= \sum_{\substack{k_1=1 \\ k_1 \neq 1}}^{n} \cdots \sum_{\substack{k_{T-1}=1 \\ k_{T-1} \neq 1}}^{n} \sum_{k_T=1}^{n} P_k P_{k_1,k_2} \cdots P_{k_{T-1},k_T} P_{K_T,j} -$$

$$- \sum_{\substack{k_1=1 \\ k_1 \neq j}}^{n} \cdots \sum_{\substack{k_{T-1}=1 \\ k_{T-1} \neq j}}^{n} P_{k_1} P_{k_1,k_2} \cdots P_{k_{T-2},k_{T-1}} P_{k_{T-1},j} P_{j,j} =$$

$$= \sum_{\substack{k_1=1 \\ k_1 \neq j}}^{n} \cdots \sum_{\substack{k_{T-1}=1 \\ k_{T-1} \neq j}}^{n} P_{k_1} P_{k_1,k_2} \cdots P_{k_{T-2},k_{T-1}} P_{k_{T-1},j} - P_{j,j} B_j(T-1)$$

where we used in the first term, that

$$\sum_{i=1}^{n} P_{k,i} P_{i,j}^{/s/} = P_{k,j}^{/s+1/} \qquad k,j=1,\ldots,n \text{ and } s \geq 1 .$$

We repeat this procedure by decomposing the sums, and at last we get:

$$B_j(T) = P_j - P_{j,j} B_j(T-1) - P_{j,j}^{/2/} \cdot B_j(T-2) - \ldots - P_{jj}^{/T/} \cdot B_j(0).$$

Let's use this expression for $B_j/T-1/$, let's subtract $B_j/T-1/$ from $B_j/T/$, and at last let's use the condition that $P_{j,j}^{/s/} = P_j$ for each s between 1 and T. So we get:

$$B_j/T/ = B_j/T-1/ - P_j B_j/T-1/ = /1-P_j/ B_j/T-1/$$

From this we can easily get, that

$$B_j/T/ = P_j/1-P_j/^T$$

by a simple induction. Then

$$N/T/ = \sum_{j=1}^{n} B_j/T/ = \sum_{j=1}^{n} P_j/1-P_j/^T = M/T/$$

and this is what we wanted to get.

Similarly the expected working set size is

$$Z/T/ = \sum_{j=1}^{n} /1-C_j/T//,$$

where $C_j/T/$ is the probability, that page j does not appear in the main memory. Then

$$C_j/T/ = \sum_{\substack{k_1=1 \\ k_1 \neq j}}^{n} \cdots \sum_{\substack{k_T=1 \\ k_T \neq j}}^{n} p_{k_1} p_{k_1 k_2} \cdots p_{k_{T-1}, k_T}$$

and by using the same method we get

$$C_j/T/ = C_j/T-1/ - B_j/T-1/$$

and from this:

$$C_j/T/ = /1-P_j/^T \quad \text{and} \quad Z/T/ = \sum_{j=1}^{n} /1-/1-P_j/^T/ = S/T/$$

so the theorem is proved.

Finally we would like to show a new result. László Kajtár, a young assistant at the Eötvös University of Budapest, examined the condition $p_{j,j}^{/k/} = p_j$ for each j=1,...,n and k=1,2,..., /We used it only for k=1,...,T /. We proved in the proof of the theorem, that if $p_{j,j}^{/k/} = P_j$ for k=1,...,T and for j=1,2,...,n, than $B_j/T/ = P_j/1-P_j/^T$. Kajtár proved, that if $B_j/T/ = P_j/1-P_j/^T$ for each j and for each T=1,2,..., than $p_{j,j}^{/k/} = P_j$ for each k.

References

[1] Fagin R., Easton M.C., The independence of miss ratio on page size, J. of ACM, 23 No.1. /1976/, 128-146.

[2] Aho A.V., Denning P.J., Ullman J.D., Principles of optimal page replacement, J. of ACM, 18, No,1. /1971/, 80-93.

An algebraic model of the distributed computer system.

Jan Rudolf Just
Warsaw Technical University
00-661 Warsaw

1. Introduction.

A distributed computer system can be considered from either the hardware viewpoint or the software viewpoint. From the hardware viewpoint it is a collection of processors connected over digital communication system. From the software viewpoint it is a distributed facility for the sharing of resources, facilities and information. Since the overall design goal of distributed computer systems is an integrated hardware and software system which satisfies certain performance requirements and design constrains, it is necessary to take into consideration both the hatdware and software viewpoint.

In our paper a method for modeling distributed computer system is defined.

In this paper we use "process" as the basic unit in our description of the system.

By a model of the process we shall mean Mazurkiewicz FC-algorithm [7] . The model of FC-algorithm, introduced by Mazurkiewicz and later discussed by others, is a convenient mathematical tool for the investigation of properties of iterative processes.

Nex due to Janicki [4] we introduce a mechanism making possible an interaction of processes.

In order to describe the set of processes, we shall introduce a mathematical object, called a matrix of coprocesses.

System presented in this paper might be mean as model for operating system working in "asynchronous parallel" where several processing units cooperate.

On the basis of this theory a method of synthesis of processes in distributed computer system, and a method for reliability analysis of multiprocessors, multiacces computer systems are given in [6] ,[1].

Problem called a synthesis of processes in distributed computer system is the solution of following question. W_e have a given computer network and given process/program/, and we want to execute this program in this network. With network are connected some constrains as a result impossibility of execution some particular part of process by a given processor. On the other hand, process execution in network has some requirements such as facilities of system.
The problem of synthesis of processes in distributed computer systems will be such a distribution of processes in the system / such an allocation of tasks to particular processors /, that execution of the particular process will be feasible take into consideration these requirements and constrains. The method given in [6] lets in a formal way to solve this problem.

And what about reliability analysis of multiprocessors system, it is possible to determine the probability of correct tasks realization in the system.

On the basis of model presented, some conditions of independence of procrsses in system are defined [5] . The independed processes are allowed to be performed concurrently.

2.Basic notations.

In this chapter there are discussed elements of the theory, important to the problem examined in this paper, taking from the basic publications described this theory / [4] , [7] ,[8] / .

Let X be a set. By Rel(X) we denote the set of $Rel(X) = \{ R \mid R \subseteq X \times X \}$ and by id we denote the identity relation in Rel(X). For each relation R , let $R^0 = id$. For $R \in Rel(X)$, $R^* = \bigcup_{i=0} R^i$, $R^+ = \bigcup_{i=1} R^i$.

Let Σ be an alphabet. A word over an alphabet is a finite sequence of elements in Σ . The word of lenght zero is denoted by ε . The set of all words, including ε , over an alphabet Σ is denoted by Σ^* . We put also $\Sigma^+ = \Sigma^* - \{\varepsilon\}$, $\Sigma' = \Sigma \cup \{\varepsilon\}$.

By a net we mean any system [7] : $N = (U, \leqslant, \circ, e, \underline{0})$, where:

1/ $(U, \leqslant)$ is complete lattice,

2/ $(U, \leqslant, e, \underline{0})$ is a monoid with unity e , and with zero $\underline{0}$,

3/ a binary relation $\circ$ is additive and continous.

By a finite-control /FC/ algorithm over N we mean any system [7] $A = (V_A, a_1, \varepsilon, P_A)$ where:

$V_A = \{a_1, a_2, \ldots, a_k\}$ is an arbitrary set of control symbols,

$a_1 \in V_A$ is the start symbol ,

ε is the terminal symbol,

P_A is a finite set of triples of the form (a_i, r, a) , where $a_i \in V_A$, $a \in V_A^*$, $r \in U$.

By I(n) we denote the set $\{1, 2, \ldots, n\}$.

3. Formulation of the problem.

A distributed computer system / DCS / is regarded as a collection of processors, located at various stations, interconnected by a communication network. This is the hardware aspect of the system.

Lets sign: $\mathcal{T}$ - the set of processors, E - the set of transmission lines.

The processor capacities are represented by the vector $\bar{d}$.

This vector expressed in parametric way such facilities as: processor throughput, main stotage capacity, maximum number of terminals attachable to the processor, etc.

Transmission lines connecting stations in DCS , are characterised by the vector $\bar{b}$. It may include such items as line speed, line buffer capacity etc. [2]

A task realization in DCS is the result of activity of distributed in system processes, connected asynchronously.

Such system can be functionally discussed as a set of interacted processes MP. $MP = \{ MP_k \mid k \in I(m) \}$.

During task realization the user of system creates so called the virtual network of processes [3] .

The virtual network of processes consists of a set of logically connected processes. We assume, that each user of DCS realize his own virtual process.

Each of these processes consists of a coprocesses set of the given virtual process.

$$MP_k = (I_0^k, A_1^k, \ldots , A_n^k)$$

where:

MP_k - k-th the virtual process in DCS, which is composed with n coprocesses $A_1^k, \ldots , A_n^k$.

$I_0^k \in I(n)$ - the number of the initial coprocess.

For a given virtual process MP, a set of coprocesses of this process create a network.

$$SMP_k = (A_i^k, KL^k) \quad , \quad i \in I(n)$$

where: $KL^k = (A_{i_1}^k \times A_{i_2}^k)$ for $i_1, i_2 \in I(n)$.

Each of coprocesses for a given virtual process is executed in a different processor of DCS.

4. Model.

By a model of distributed computer system we shall mean any system:

$$DCS = (S, MP, \mathcal{K})$$

where:

S - a structure of DCS , / see 4.1/

MP - a set of processes in DCS, / See 4.2 /

$\mathcal{K}$ - an allocation function $\mathcal{K} : MP \rightarrow S$ / see 4.3/.

4,1. Structure of DCS.

By a structure of DCS we mean a graph:

$$S = (N, n_0, LT)$$

where:

N - a set of nodes of network,

$n_0 \in N$ - an initial node,

$LT \subseteq N \times N$ - a set of transmission lines.

Elements of the real DCS we can assign to abstract elaments of the model, determining functions: / see 2 /

$\varnothing_p : N \rightarrow \Pi$ processor allocation function,

$\varnothing_e : N \times N \rightarrow E$ lines allocation function.

4.2. Processes in DCS.

A task realization in DCS is the result of activity of distributed processes.

In order to describe the set of processes in DCS we shall introduce a mathematical object, called a matrix of coprocesses.

4.2.1. Matrix of coprocesses.

Let $N = (U, \leqslant, \circ, e, \underline{Q})$ be an arbitrary net [7] .

By a matrix of coprocesses over N we mean any system:

$$MP = (A(MP) , [i'_0, i_0])$$

where:

$$A(MP) = \begin{bmatrix} A_{11} , A_{12} , \dots , A_{1n} \\ \dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots \\ A_{m1} , A_{m2} , \dots , A_{mn} \end{bmatrix}$$

Each line in the above matrix represent one process.

The pair $[i'_0 , i_0]$ indicate the start process,

$i'_0 \in I(m)$ - a number of the process,

$i_0 \in I(n)$ - a number of a component of process / coprocess/,

$A_{i'i}$ - a triple wchich represent the i-th coprocess in the i'-th process.

$$A_{i'i} = (V_{i'i}, \sigma_{i'i}, P_{i'i})$$

where:

/1/ $V_{i'i}$ - an alphabet / of control symbols of $A_{i'i}$ /,

/2/ $\sigma_{i'i} \in V_{i'i}$ - the start symbol of $A_{i'i}$,

/3/ $P_{i'i}$ - a finite subset of the set:

$$\{ (i',i) \times (I(m) \times I(n)) \times (V \times V^*) \times U \} .$$

This mean that $P_{i'i}$ is a finite set of 4-tuples of the form:

$$(i' \to j', i \to j, a \to b, R) \quad \text{where:}$$

1/ $i' \to j' \in \{i'\} \times I(m)$,

2/ $i \to j \in \{i\} \times I(n)$,

3/ $a \rightarrow b \in V \times V^{\bullet}$,

4/ $R \in U$.

$P_{i'i}$ is called the set of instructions of $A_{i'i}$.

The empty coprocess of the form $(\{\varepsilon\}, \varepsilon, \emptyset)$ will be denoted by Θ.

Each instruction consists of four parts:

a/ $i' \rightarrow j'$ indicates the process which will be active after the execution of instruction / j'-th process will be active /,

b/ $i \rightarrow j$ indicates the coprocess which will be active after execution of instruction,

c/ $a \rightarrow b$ indicates the current /a/ and next /b/ point of the $A_{i'i}$ component,

d/ R the "action" of the instruction.

R - in respect to abstract character, we shall mean as the program, the part of the program or activity of the operating system.

Let $E_{i'i} = \{ R \in U \mid \exists (i' \rightarrow j', i \rightarrow j, a \rightarrow b, R) \in P_{i'i} \}$ for $i' \in I(m)$, $i \in I(n)$.

The set of "actions" of the matrix MP we denote by E_M.

$$E_M = \bigcup_{i'=1}^{m} \bigcup_{i=1}^{n} E_{i'i} .$$

Let:

$$ms = \begin{bmatrix} V^{\bullet}_{11}, & V^{\bullet}_{12}, & \cdots, & V^{\bullet}_{1n} \\ \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ V^{\bullet}_{m1}, & V^{\bullet}_{m2}, & \cdots, & V^{\bullet}_{mn} \end{bmatrix}$$

The set $MS = \{ I(m) \times I(n) \times [\![ms]\!] \}$ is called the set of control states of rhe matrix MP, where $[\![\ ]\!]$ is a function:

$$[\![\ ]\!] : \begin{bmatrix} V_{11}, & V_{12}, & \cdots, & V_{1n} \\ \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ V_{m1}, & V_{m2}, & \cdots, & V_{mn} \end{bmatrix} \longrightarrow V_{11} \times V_{12} \times \cdots \times V_{1n} \times V_{21} \times \cdots \times V_{m-1n} \times V_{m1} \times \cdots \times V_{mn}$$

Let coord: $MS \times I(m) \times I(n) \rightarrow \bigcup_{i'=1}^{m} \bigcup_{i=1}^{n} V_{i'i}$ be a function such thet:

$y = (i', i, V_{11}, V_{12}, \ldots, V_{mn}) \in MS$ implies $\forall k': 1 \leq k' \leq m$, $\forall k: 1 \leq k \leq n$ coord $(MS, k', k) = V_{k'k}$.

Each $(i' \to j',\ i \to j,\ a \to b,\ R)$ can be regarded as a relation in MS, defined in the following way:

if $y_1\ (i' \to j',\ i \to j,\ a \to b,\ R)\ y_2$ and coord $(y_1, i', i) = a$

then coord $(y_2, i', i) = b$, $y_1, y_2 \in MS$.

The set $MT = \{ [\![ms]\!] \times I(m) \times I(n) \in MS \mid \text{coord}(MS, i', i) = \varepsilon \}$ $i' \in I(m)$, $i \in I(n)$, is called the set of terminal control states of MP .

The set of instructions of the matrix MP is the set:

$$P_M = \bigcup_{i'=1}^{m} \bigcup_{i=1}^{n} P_{i'i} \ .$$

Consider now a finite sequence of elements P_M , such that there exists a sequence of control states: $y_1, \ldots, y_{m+1}$ with the following properties:

1/ $\forall (k \leq d)$ $y_k\ (i'_k \to j'_k,\ i_k \to j_k,\ a_k \to b_k,\ R_k)\ y_{k+1}$

2/ $y_{d+1} \in MT$.

To above sequence of control states corresponds the sequence of "actions": $(R_1, \ldots, R_d)$.

The finite set of all sequences of control states we denote by $Tr_M(y)$.

Let $Tr(M) = \bigcup \{ Tr_M(y) \mid y \in MS \}$,

and function $\mathcal{H}$: $Tr(M) \to U$ let be defined by:

$$\mathcal{H}((R_1, \ldots, R_d)) = R_1 \circ R_2 \circ \ldots \circ R_d$$

for $\forall (R_1, \ldots, R_d) \in Tr(M)$.

Each element from Tr(y) / trace/ produce its own outcome therefore the outcome of the whole set will be the join

$$Tail_M(y) = \bigcup \{ \mathcal{H}(t) \mid t \in Tr(M) \} \ .$$

The concept of Tail- element was introduced originally by Mazurkiewicz for sequential processes [8].

In our model the vector $Tail_M(y)$ expressed outcome of the virtual process in DCS .

Matrix of coprocesses is useful where the number of processes and coprocesses is finite and known, and processes are iterative. Example. Consider the system which consists of two processes such that the first process consists of two coprocess. Let this system be represented by the flowdiagram. / Fig.1 /

The interpretation of the above flowdiagram is the following.
Single arrows indicate the next point of coprocess.
Double arrows indicate the reactivation point of coprocess.
Thick arrows indicate the reactivation point of process.

O - the reactivation point of coprocess. ⊙ - the reactivation point of process.

Let $N = (\mathrm{Rel}(X), \subseteq, \circ, id, \emptyset)$ be the net of binary relations
In this case the matrix of coprocesses can be defined as follows:

$MP = (A(MP), [1,1])$ where:

$$A(MP) = \begin{bmatrix} A_{11}, A_{12} \\ A_{21}, \Theta \end{bmatrix}$$

where: $A_{11} = (\{\sigma_{11}, a_{11}, a_{12}\}, \sigma_{11}, P_{11})$

$P_{11} = \{(1\to1,\ 1\to1,\ \sigma_{11}\Rightarrow a_{11},\ R_{11}),\ (1\to1,\ 1\to1,\ a_{11}\Rightarrow a_{12},\ R_{12}),$
$(1\to1,\ 1\to2,\ a_{12}\Rightarrow a_{11},\ id)\}$

$A_{12} = (\{\sigma_{12}, b_{11}, b_{12}, b_{13}\}, \sigma_{12}, P_{12})$

$P_{12} = \{(1\to1,\ 2\to2,\ \sigma_{12}\Rightarrow b_{11},\ S_{11}),\ (1\to1,\ 2\to2,\ b_{11}\Rightarrow b_{12},\ S_{12}),$
$(1\to2,\ 2\to2,\ b_{12}\to\sigma_{12},\ id),\ (1\to1,\ 2\to2,\ b_{12}\Rightarrow b_{13},\ S_{13}),$
$(1\to1,\ 2\to1,\ b_{13}\Rightarrow b_{11},\ id)\}$

$A_{21} = (\{\sigma_{21}, a_{21}, a_{23}\}, \sigma_{21}, P_{21})$

$P_{21} = \{(2\to2,\ 1\to1,\ \sigma_{21}\Rightarrow a_{21},\ R_{21}),\ (2\to2,\ 1\to1,\ a_{21}\Rightarrow \varepsilon,\ R_{22}),$
$(2\to2,\ 1\to1,\ a_{21}\Rightarrow a_{23},\ R_{23}),\ (2\to1,\ 1\to1,\ a_{23}\Rightarrow \sigma_{21},\ id)\}$

In this example:

$E_M = E_{11} \cup E_{12} \cup E_{21} = \{R_{11}, R_{12}, S_{12}, S_{11}, S_{13}, R_{21}, R_{22}, R_{23}, id\}$.

$MS = \{1,2\} \times \{1,2\} \times \{\sigma_{11}, a_{11}, a_{12}\} \times \{\sigma_{12}, b_{11}, b_{12}, b_{13}\} \times \{\sigma_{21}, a_{21}, a_{23}, \varepsilon\} \cup \{\emptyset\}$

$[\![x_0]\!] = (\sigma_{11}, \sigma_{12}, \sigma_{21}, \varepsilon)$.

4.2.2. Semantics of matrix of coprocesses.

Proving properties of system of processes / in our model / is proving properties of vector $Tail_M$. Thus this vector must be found explicitly. It is possible by proving that $Tail_M$ is the least solution of so-called canonical set of equations for each given matrix of coprocesses.

Analysis of vector Tail and the method of solving the canonical set of equation are adopted from the theory [4] , [7] , [8] .

Let MSP will be called proper control states of MP.

$$MSP = \bigcup_{i'=1}^{m} \bigcup_{i=1}^{n} MSP_{i'i} \quad , \text{ where:}$$

$$MSP_{i'i} = \{i'\} \times \{i\} \times MR_{11} \; MR_{12} \times \ldots \times MR_{i'i-1} \times V^{\bullet}_{i'i} \times MR_{i'i+1} \ldots \times MR_{mn}$$

$$MR_{i'i} = \{ b_{i'i} \in V^{\bullet}_{i'i} \mid \exists (i' \to j', i \to j, a \to b, R) \in P_{i'i} \text{ and } ((i' \neq j') \vee (i \neq j)) \} \cup \{\sigma_{i'i}\}$$

where: $(i',i) \neq (i'_0, i_0)$ and

$$MR_{i'_0 i_0} = \{ b_{i'i} \in V^{\bullet}_{i'_0 i} \mid \exists (i'_0 \to j', i_0 \to j, a \to b, R) \in P_{i'_0 i} \text{ and } (i'_0, i_0) \neq (j', j) \} .$$

For every set X by $seq(X)$ we denote the set of all sequences of elements of X. Let $t \in seq(X)$. We shall write $a \sqsubset t$, if $t = (t_1, a, t_2)$ where $a \in X$ and $t_1, t_2 \in seq(X)$.

Lemma. If $z \in MSP$ then $\{ y \mid y \sqsubset z\text{-state trace} \} \subseteq MSP$. ■

We can see that $x_0 = (i'_0, i_0, [\![ms_0]\!]) \in MSP$, where:

$$ms_0 = \begin{bmatrix} \sigma_{11}, \sigma_{12}, \ldots, \sigma_{1n} \\ \ldots\ldots\ldots\ldots \\ \sigma_{m1}, \sigma_{m2}, \ldots, \sigma_{mn} \end{bmatrix}$$

From this and lemma, it follows that all elements of all x_0-state-trace belong to MSP.

Now we formulate the fixpoint semantics of the matrix MP.

Let : $\mathcal{F} : MSP \times MSP \to U$,

$\mathcal{G} : MSP \to U$

be functions, defined in the following way:

$$\mathcal{F}((i',i,a_{11},a_{12},\ \dots\ ,a_{mn}),\ (j',j,b_{11},b_{12},\ \dots\ ,b_{mn})) =$$

$$= \begin{cases} \{R \mid \exists (i' \to j', i \to j, a_{i'i} \to b_{i'i}, R) \in P_{i'i} \text{ if } b_{k'k}=a_{k'k} \text{ for } k' \in I(m), \\ \qquad k \in I(n) \text{ and } (k' \neq i' \vee k \neq i) \\ \underline{0} \text{ - otherwice} \end{cases}$$

$$\mathcal{G}((i',i,a_{11},a_{12},\ \dots\ ,a_{mn})) = \begin{cases} e & \text{for } a_{i'i}=\varepsilon \\ \underline{0} & \text{for } a_{i'i}\neq\varepsilon \end{cases}$$

Since $d=card(MSP)$, we can put $MSP = \{z_1, \dots, z_d\}$ where $z_1=x_0$.

We can write $(X_{z_1}, \dots, X_{z_d}) \in U^d$ instead of $(X_1, \dots, X_d) \in U^d$.

The canonical set of equation of matrix of coprocesses is following:

$$X_{z_1} = \bigcup_{z \in MSP} \mathcal{F}(z_1,z) \cdot X_z \cup \mathcal{G}(z_1)$$

$$\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots$$

$$X_{z_d} = \bigcup_{z \in MSP} \mathcal{F}(z_d,z) \cdot X_z \cup \mathcal{G}(z_d)$$

Theorem. For every matrix of coprocesses MP the vector

$$(Tail_M(z_1), \dots, Tail_M(z_d))$$

is the least solution of the canonical set of equations of MP. ■

Example/continuation/.

The solution of the canonical set of equation of given matrix MP is:

$$Tail_M(x_0) = R_{11}R_{12}S_{11}S_{12}((R_{21}R_{23} \cup S_{13}R_{12})S_{12})^*R_{21}R_{22}.$$

The set of equations of given matrix of coprocesses is equivalent to the flowchart from Fig.2.

4.3. Definition of the function $\mathcal{K}$.

The function $\mathcal{K}$ is the third element of the DCS model. To describe a particular system, it is necessary to specify:

1/ how to allocate processes to processors,

2/ how to allocate communication lines between that processors.

It is realized by function $\mathcal{K}$.

The structure of DCS we was defined as: $S = (N, n_0, LT)$. Processes in DCS creates the virtual process graph G / details in [5] /. This graph can be expressed in following way: $G = (MS, PS)$, where $PS \subseteq MS \times MS$.

The function $\mathcal{K}$ is a some homomorphism between Structure of DCS and the graph of given virtual process G.

Its mean, that the structure of logical channels, between the components of the given process, must be adequate to the structure of connections between processors of DCS.

Specifing the function $\mathcal{K}$ we must take into consideration the "technical conditions" of DCS, and requirements of the executed process components.

References.

1. Bieńkowski K., Just J.R., Reliability models of multiacces, multiprocessors computer systems.Relcomex 79 , Książ ,Poland.
2. Chang S.K., A model for distributed computer system design. IEEE Trans.on System, Man and Cybernetics, vol.SMC-5,no.6,1976.
3. Głuszkow W.M., Sieci EWM, Swjaz,Moskwa,1977.
4. Janicki R., Vector of coroutines over Blikle nets.Lecture Notes in Computer Science, vol.56, Springer-Verlag,pp.113-119,1977.
5. Just J.R., An analysis of processes in distributed computer network. / to appear/.
6. Just J.R., The synthesis of processes in computer networks. / to appear/.
7. Mazurkiewicz A., Pawlak Z., Mathematical Foundations of Computer

Soience. PWN Warsaw, 1977.

8. Mazurkiewicz A., Proving algorithms by tail functions. Information and control, no.18, pp. 220-226, 1971.

Acknowledegements

I would like to think Dr. R. Janicki from Institute of Mathematics Warsaw Technical University for his suggestions and criticism.

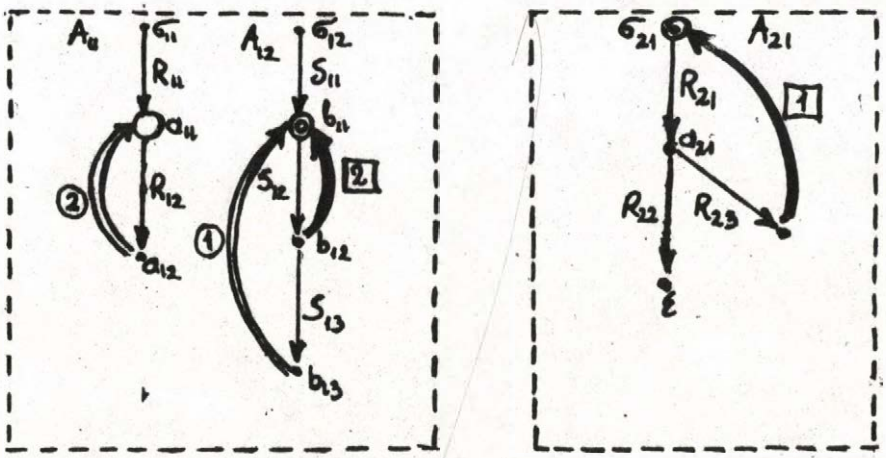


Fig. 1.

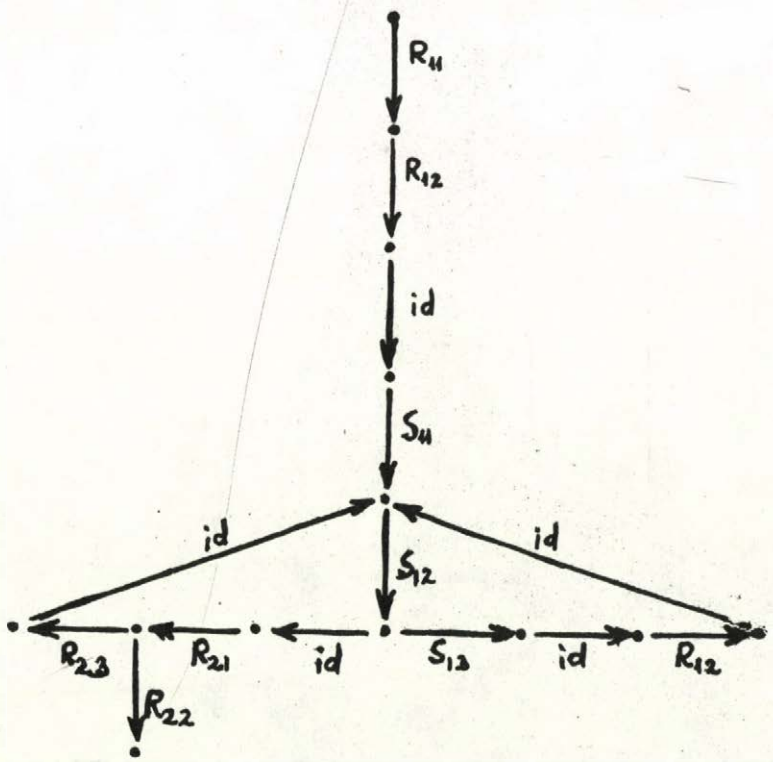


Fig. 2.

A Scheme for Conceiving Methods to Control the Degree of Multiprogramming for Virtual Memory Computer Systems

C. Glowacki

Universite Libre de Bruxelles Faculte des Sciences
Laboratoire D'Informatique Theorique

A b s t r a c t

We present a queueing network model of virtual memory computer system which is a particular case of Jackson's network. The purpose is to examine the effect of system parameters core memory size, input-output devices characteristics, ... and program behaviour parameters total compute time, input-output rates, ... on system throughout.
We make remark about methods proposed by J.A. Shils and J. Leroudier and D. Potier. We also put forward a principle for conceiving methods to control the loading of virtual memory computer systems.

1. INTRODUCTION

Several authors have applied queueing network to the analysis of computer system performance [1,3,5,6,8]. In this paper, to examine the effect of sharing of memory between programs, and hence of the degree of multiprogramming, on the system throughput, we characterize the computer system random variables as functions of program behaviour parameters.

From representative examples, we show that the secondary memory device activity of a well-working system is similar to the whole system activity.
We also point out some lacunas in the method proposed by A.J. Shils [9] and make remarks on the algorithm presented by J. Leroudier and D. Potier [8].
Finally, we present a scheme for conceiving methods to control the degree of multiprogramming for virtual memory computer systems.

2. THE MODEL

Consider the system described in figure 1. It represents the system architecture of a multiprogrammed paged virtual memory computer.
It consists of a central processor unit (CPU), a secondary memory device and p input-output devices. The behaviour of programs is characterized by a compute time followed by either a departure from the system or a page fault or an input-output request. In these last two states, the program enters in an appropriate queue associated with the concerned device. Programs which terminate their service at the input-output device return to the CPU queue.
Let

t_{CPU} be the random variable representing ininterrupted compute time at the CPU,

t_{SM} be the random variable representing the compute time between two successive page faults,

t_{I/O_k} be the random variable representing the compute time between two successive I/O requests for input-output device k, k=1,...,p,

t_C be the random variable representing total compute time of a program.

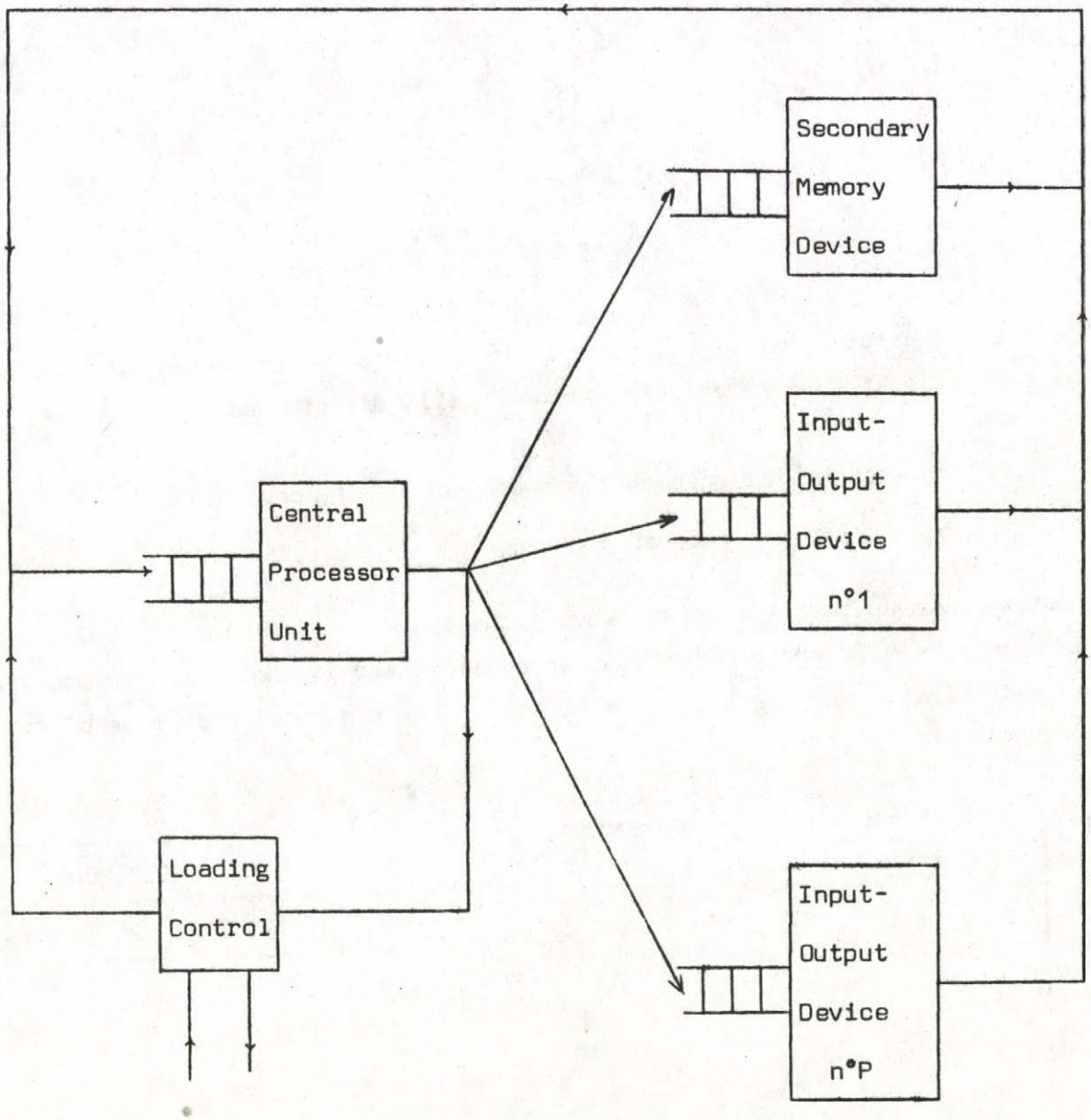


Figure 1

We assume that t_{SM}, t_C and t_{I/O_k} are independent and exponentially distributed.

Let μ_{CPU}, μ_{SM}, μ_{I/O_k} and μ_C be given by

$$\mu_{CPU} = E^{-1}(t_{CPU})$$

$$\mu_{SM} = E^{-1}(t_{SM})$$

$$\mu_{I/O_k} = E^{-1}(t_{I/O_k}) \qquad k = 1,\ldots,p$$

$$\mu_C = E^{-1}(t_C) \ .$$

Moreover we assume that the service times at the secondary memory device and at the input-output devices are also exponentially distributed.

Let

λ_{SM} be the service rate at the secondary memory device,

λ_{I/O_k} be the service rate at input-output device k, k=1, ..., p.

First we suppose that a program leaving the system is immediately replaced. The degree of multiprogramming is then constant and in that case the model may be schematically represented as in figure 2. It is a particular case of Jackson's networks [7].

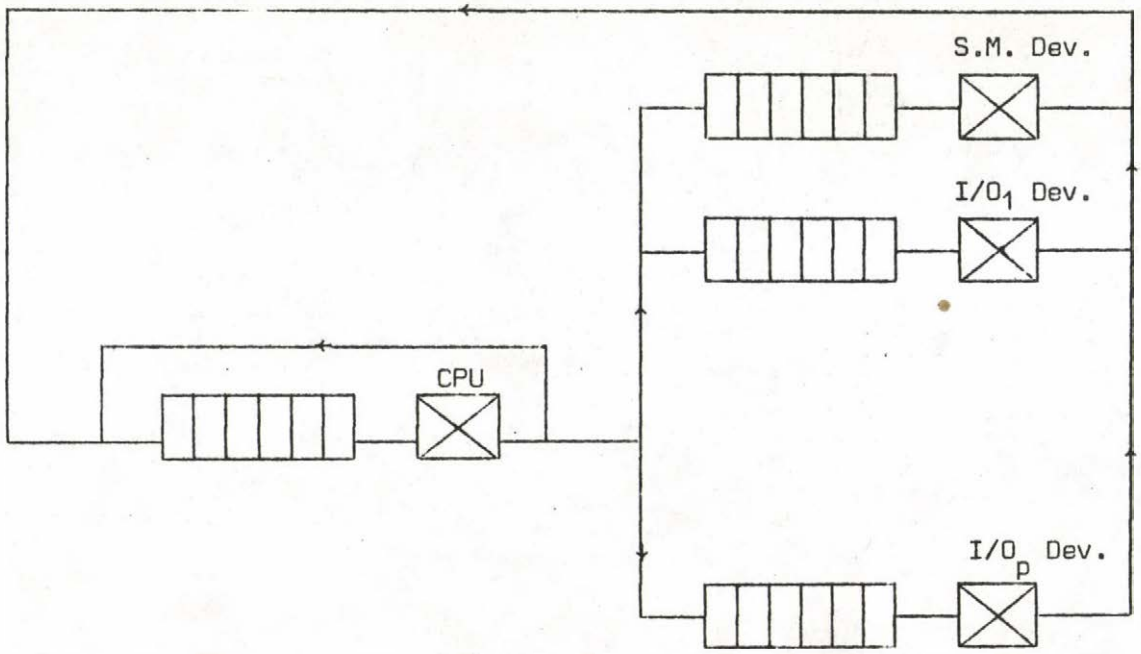


Figure 2

Let us define

$$\rho_{SM} = \frac{\mu_{SM}}{\lambda_{SM}}$$

$$\rho_{I/O_k} = \frac{\mu_{I/O_k}}{\lambda_{I/O_k}} \qquad k=1,\ldots,p$$

respectively the utilization index number of the secondary memory device and of the input-output devices.

It now remains to introduce the effect of memory sharing. Experimental evidence indicates that the mean time between page faults μ_{SM}^{-1} is an increasing function of the amount of space allocated to each program. Furthermore we shall assume that core memory available is equally shared among the loaded programs and that the other parameters are independent of the degree of multiprogramming. It results that ρ_{SM} is an increasing function of the degree of multiprogramming. Thus the optimal degree of multiprogramming depends on the different input-output index numbers and on function $\rho_{SM}(n)$, where n is the degree of multiprogramming.

Here we only analyze balanced systems i.e. we have

$$\rho_{I/O_k} = \rho_{I/O} \qquad k=1,\ldots,p$$

The CPU utilization rate is given by

$$A_{CPU} = \frac{\frac{1-\rho_{I/O}^{n+1}}{1-\rho_{I/O}} - \frac{1-\rho_{SM}^{n+1}}{1-\rho_{SM}}}{\frac{1-\rho_{I/O}^{n+2}}{1-\rho_{I/O}} - \frac{1-\rho_{SM}^{n+2}}{1-\rho_{SM}}} \qquad (1)$$

It is interesting to observe that A_{CPU} is independent of the devices number.

Now we suppose that n is a real number.

When we compute the first derivative of A_{CPU}, we obtain

$$A'_{CPU} = \frac{(\rho_{I/O}-\rho_{SM})^2}{(\rho_{I/O}a_{I/O}-\rho_{SM}a_{SM})^2} \times \left\{\frac{a_{SM}\cdot g_{I/O}-a_{I/O}\cdot g_{SM}}{\rho_{I/O}-\rho_{SM}} + \frac{\rho'_{SM}}{(\rho_{SM}-1)}\left[\frac{a_{SM}\cdot a_{SM}^{I/O}-(n+1)\rho_{SM}^{n}\cdot a_{I/O}}{(\rho_{I/O}-\rho_{SM})}\right]\right\} \qquad (2)$$

where

$$a_{SM} = \frac{1-\rho_{SM}^{n+1}}{1-\rho_{SM}} \qquad a_{I/O} = \frac{1-\rho_{I/O}^{n+1}}{1-\rho_{I/O}}$$

$$g_{SM} = \frac{\rho_{SM}^{n+1} \cdot \ell n \rho_{SM}}{\rho_{SM}^{-1}} \qquad g_{I/O} = \frac{\rho_{I/O}^{n+1} \cdot \ell n \rho_{I/O}}{\rho_{I/O}^{-1}}$$

$$a_{SM}^{I/O} = \frac{\rho_{I/O}^{n+1} - \rho_{SM}^{n+1}}{\rho_{I/O} - \rho_{SM}}$$

It is easy to demonstrate that the optimal degree of multiprogramming annuls the following function

$$\frac{g_{SM} \cdot a_{I/O} - g_{I/O} \cdot a_{SM}}{\rho_{SM} - \rho_{I/O}} + \frac{\rho'_{SM}}{\rho_{SM}^{-1}} \left[\frac{(n+1)\rho_{SM}^{n} \cdot a_{I/O} - a_{SM} \cdot a_{SM}^{I/O}}{\rho_{SM} - \rho_{I/O}} \right] \qquad (3)$$

We will use the function proposed by Belady and Kuehner [2] to relate the mean time μ_{SM}^{-1} between two consecutive page faults of a program to the amount of space allocated to this program.

It results that for ρ_{SM}, we have the following expression

$$\rho_{SM} = \frac{n^k}{\lambda_{SM} \cdot a \cdot M^k} \qquad (4)$$

where

n is the degree of multiprogramming,

M is the total memory size,

a is a parameter depending on processing speed as well as on program characteristics,

k is a parameter depending on program locality and memory strategy.

In that case, function (3) becomes

$$\frac{g_{SM} \cdot a_{I/O} - g_{I/O} \cdot a_{SM}}{\rho_{SM} - \rho_{I/O}} + \frac{k \cdot \rho_{SM}}{n(\rho_{SM}^{-1})} \cdot \left[\frac{(n+1)\rho_{SM}^{n} \cdot a_{I/O} - a_{SM} \cdot a_{SM}^{I/O}}{\rho_{SM} - \rho_{I/O}} \right] \qquad (5)$$

3. ANALYSIS OF THE INFLUENCE OF SOME PARAMETERS ON THE THROUGHPUT

Let us examine the behaviour of systems on the following concrete model. Consider a balanced system having

- a secondary memory device of which the mean service time is 10 ms,
- a set of input-output devices,
- 512 pages of core memory.

We use the function of Belady and Kuehner to introduce the effect of memory sharing.

Thus, we have

$$\rho_{SM} = \frac{n^k}{100a(512)^k}$$

Let us observe the influence of parameters a, k and $\rho_{I/O}$ on the working of the system. We only examine the model for values of parameters verifying the following inequality

$$\rho_{SM}(n=2) < \max(1, \rho_{I/O})$$

This inequality means that we assume that the system is not satured when two programs are loaded in the core memory. In the below examples, we use the values of parameters presented in table 1. This set is a representative sample.

k	1.8	2.0	2.2		
a	10^{-6}	2.10^{-6}	5.10^{-6}		
$\rho_{I/O}$	0.3	0.6	1.2	2.4	4.8

Table 1

Table 2 presents

- the CPU utilization rate
- the pagination index number
- the degree of multiprogramming

when the working is optimal for the considered parameters values.

		$a=0.1\times10^{-5}$			$a=0.2\times10^{-5}$			$a=0.5\times10^{-5}$		
	k	A_{CPU}	ρ^*_{SM}	n^*	A_{CPU}	ρ^*_{SM}	n^*	A_{CPU}	ρ^*_{SM}	n^*
$\rho_{I/O}=0.3$	1.8	0.80	0.47	2.0	0.90	0.42	2.8	0.98	0.38	4.4
	2.0	0.94	0.37	3.1	0.98	0.35	4.3	0.99	0.33	6.6
	2.2	0.99	0.33	4.6	0.99	0.32	4.4	0.99	0.31	9.5
$\rho_{I/O}=0.6$	1.8	0.72	0.61	2.3	0.83	0.56	3.3	0.93	0.53	5.3
	2.0	0.87	0.51	3.7	0.93	0.50	5.1	0.98	0.50	8.1
	2.2	0.95	0.48	5.5	0.98	0.48	7.7	0.99	0.50	11.8
$\rho_{I/O}=1.2$	1.8	0.57	0.82	2.8	0.69	0.77	3.9	0.75	0.74	8.3
	2.0	0.68	0.72	4.3	0.74	0.71	6.1	0.74	0.79	9.8
	2.2	0.75	0.69	6.6	0.79	0.71	9.1	0.75	0.82	14.2
$\rho_{I/O}=2.4$	1.8	0.37	1.13	3.3	0.40	1.06	4.7	0.41	1.02	7.6
	2.0	0.41	0.99	5.1	0.41	0.97	7.1	0.42	0.97	11.3
	2.2	0.42	0.93	7.5	0.42	0.94	10.3	0.42	0.95	15.8
$\rho_{I/O}=4.8$	1.8	0.20	1.58	3.9	0.21	1.47	5.6	0.21	1.35	8.9
	2.0	0.21	1.34	5.9	0.21	1.28	8.2	0.21	1.21	12.6
	2.2	0.21	1.20	8.5	0.21	1.16	11.4	0.21	1.12	17.0

Table 2

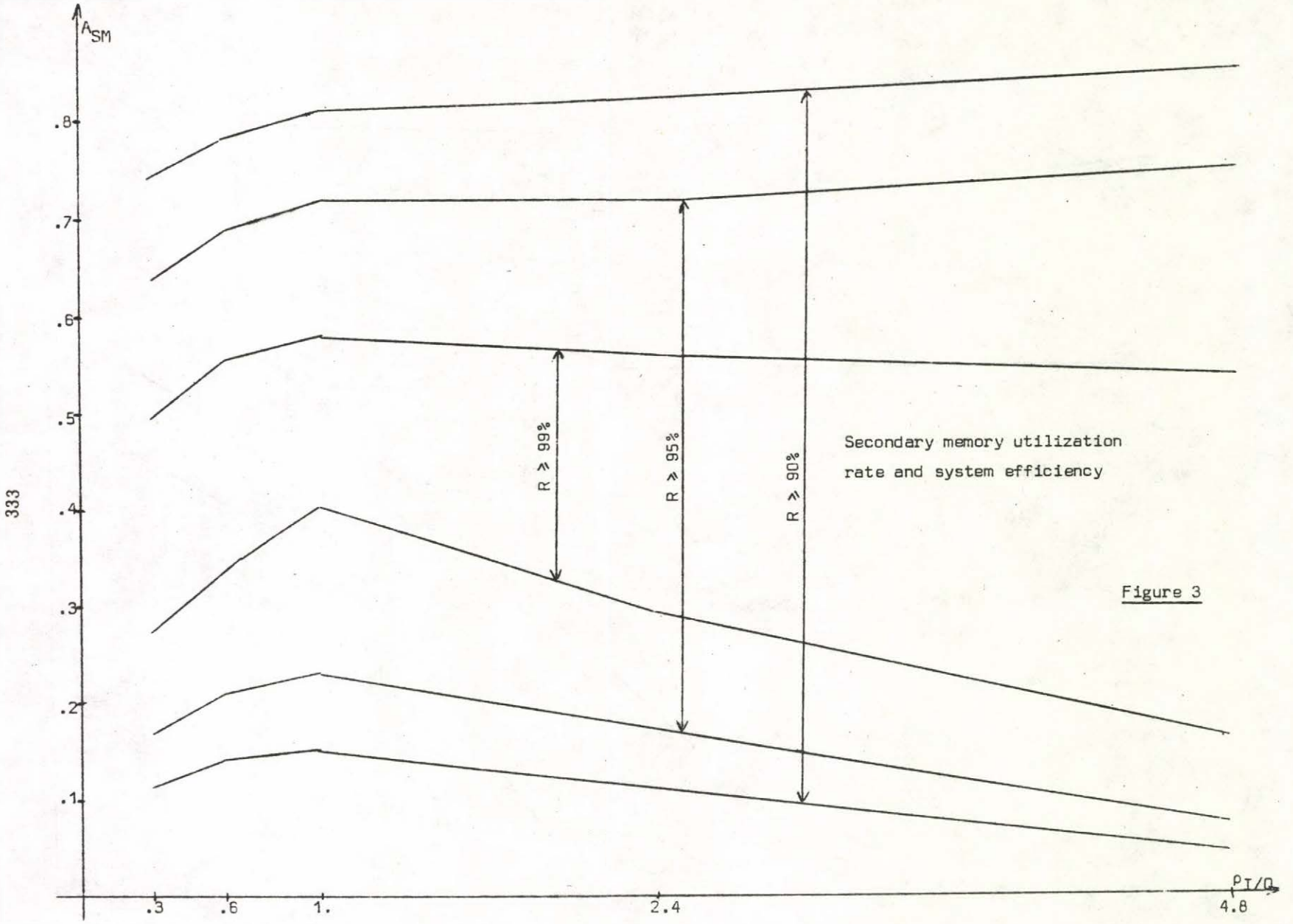
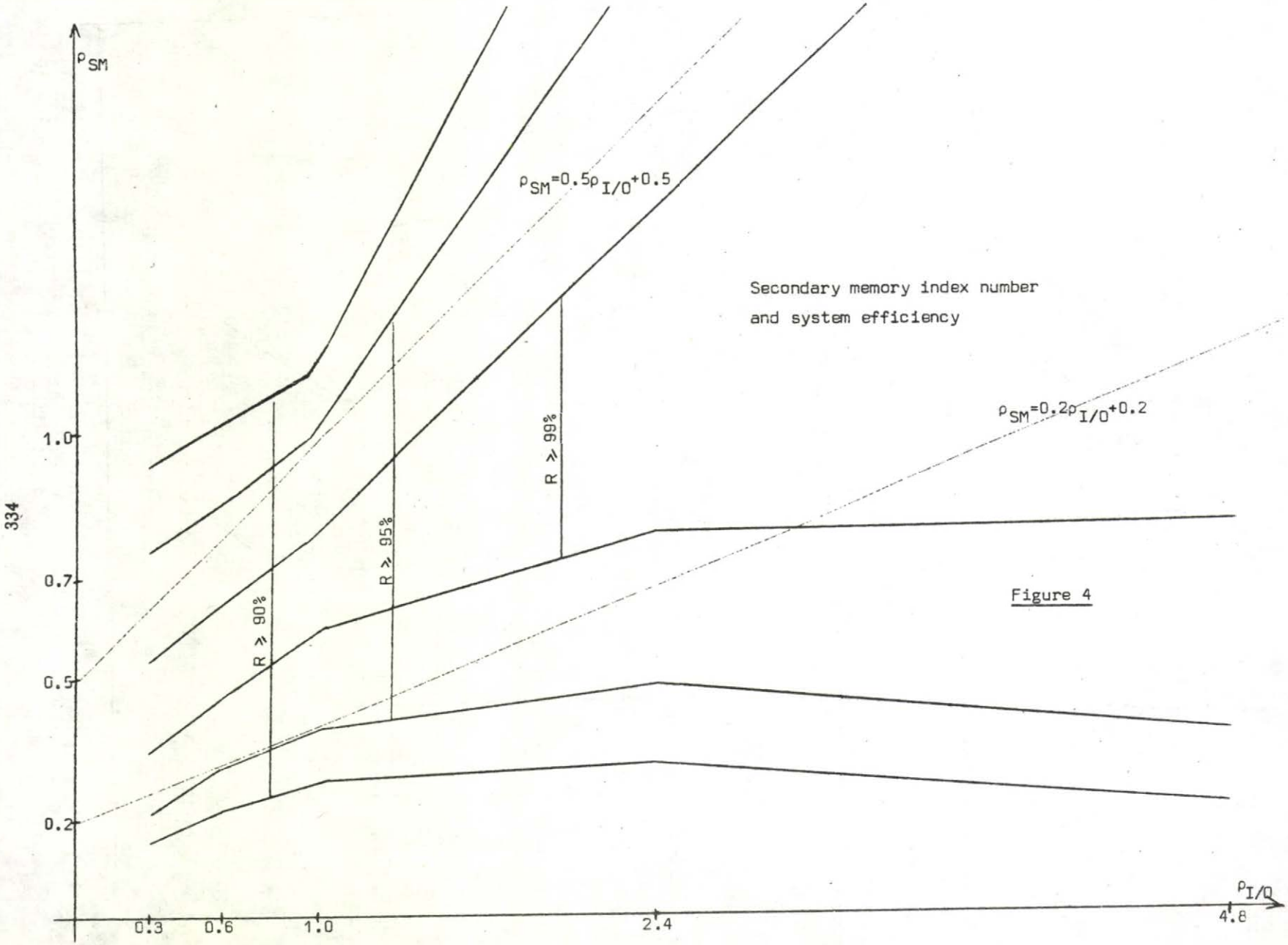

A_{SM}
.8
.7
.6
.5
.4
.3
.2
.1
R ≥ 99%
R ≥ 95%
R ≥ 90%
Secondary memory utilization
rate and system efficiency
$\rho_{I/O}$
.3
.6
1.
2.4
4.8

Figure 3



Secondary memory index number
and system efficiency

Figure 4

Let us observe that k has great part on the throughput when $\rho_{I/O}$ and the degree of multiprogramming are small. As we know that k represents in particular the efficiency of the replacement algorithm, it results that its well-choice is only important when the system bottleneck is the core memory.

Let A^n_{CPU} be the CPU utilization rate when the degree of multiprogramming is n. Let us define the system efficiency for the degree n(a, k and $\rho_{I/O}$ being fixed)

$$R(n;a,k,\rho_{I/O}) = \frac{A^n_{CPU}(a,k,\rho_{I/O})}{A^{n*}_{CPU}(a,k,\rho_{I/O})}$$

where n* is the optimal degree of multiprogramming for this model.

In figures 3 and 4, we give for each $\rho_{I/O}$ intervals where the system efficiency is greater than 90%, 95% and 99%. In figure 3, we observe that the secondary memory utilization rate remains in the 50% range when the system is well-balanced ($\rho_{I/O} \simeq 1$) and has an efficiency greater than 95%. This characteristic was observed by J. Leroudier and D. Potier [8]. Let us observe that it is not true for small $\rho_{I/O}$.

Figure 4 shows that the optimal pagination index number is an increasing function of $\rho_{I/O}$ and that the system efficiency is high ($\geq 95\%$) when the pagination index number verifies the following inequations

$$0.2\rho_{I/O}+0.2 \leq \rho_{SM} \leq 0.5\rho_{I/O}+0.5$$

As we have

$$A_{SM} = \rho_{SM} A_{CPU}$$

$$A_{I/O} = \rho_{I/O} A_{CPU}$$

we may also write

$$0.4\left(\frac{A_{CPU}+A_{I/O}}{2}\right) \leq A_{SM} \leq \frac{A_{CPU}+A_{I/O}}{2} \qquad (6)$$

These can be a criterion to control the loading.

Let us remark that $0.5(A_{CPU}+A_{I/O})$ may be regarded as a global activity measure and A_{PG} as a secondary memory device activity measure.
Thus inequations (6) mean that a well-working system has a secondary memory device activity comparable with the global system activity i.e. that it may be neither underused nor the bottleneck of the system.
It results that a well-loading of programs in a virtual memory system is achieved when the secondary memory device has an activity which is in accordance with all the system.
In [5], we arrived at the same conclusions from other models and the condition to have a well-working system was

$$A_{SM} \lesssim \max(A_{CPU}, A_{I/O_1}, \ldots, A_{I/O_p})$$

Remarks

1. In the experimental system M44/44X, A.J. Shils [9] controls the loading by observing the activity of the CPU and the secondary memory device.
   Figure 5 shows how the system activity is determined.
   There are three activity levels
   - a level of normal load characterized by a high activity of the CPU,
   - a level of underload characterized by a low level of pagination and a low CPU activity,
   - a level of overload characterized by a high level of page faults and a low activity of the CPU.

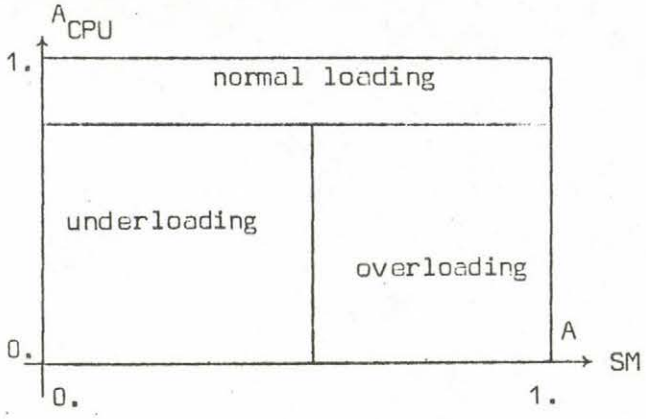


Figure 5

These levels are determined by the way of experiences.

Consider a situation where the input-output devices are the bottleneck of the system ($\rho_{I/O}>>>1$); we know that, in that case, the CPU activity is very low at the optimal loading (see table 2). Thus, for Shils, the system is either underloaded or overloaded. It results that Shils'method is unstable in this situation.

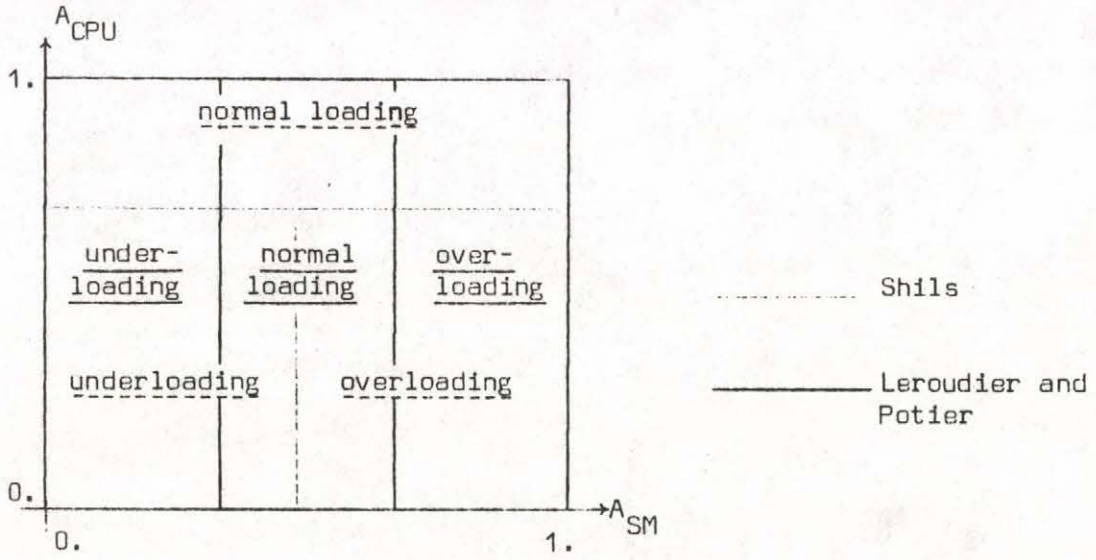


Figure 6

Figure 8 compares Shils'method and the technic of J. Leroudier and D. Potier [8]. It is easy to observe that Shils'method is inconsistant with the method of Leroudier and Potier.

2. Another way to express our criterion of controlling the loading is

$$A_{SM} \approx 0.35(A_{CPU}+A_{I/O})$$

It is easy to see that this is a generalization of the criterion of J. Leroudier and D. Potier which is

$$A_{SM} \approx 0.5$$

The proposed method is more precise because it takes the input-output operations into account.

4. SCHEME FOR CONCEIVING LOADING CONTROL METHODS

In the last paragraph, we concluded that a well-working system is obtained when the secondary memory device has an activity which is in accordance with the whole system. This fact can be used to conceive loading control methods. We must

- first define $\mu_{gl.\ act.}$ an index number based, for example, on the activity of some or all the channels, which measures the global system activity,
- then define $\mu_{sm\ act.}$ an index number of the secondary memory device activity based on the page faults rate or the secondary memory utilization rate,
- finally define how to compare them.

The loading level must be determine by the following way

$\mu_{glob.\ act.} \approx \mu_{SM\ act.}$	normal loading
$\mu_{glob.\ act.} < \mu_{SM\ act.}$	overloading
$\mu_{glob.\ act.} > \mu_{SM\ act.}$	underloading

Let us note that we can imagine very sophisticated index numbers which may be computed outside the computer by a microprocessor having leads in the concerned channels to reduce the system overheads.

5. CONCLUSION

We have presented in this paper a scheme for conceiving loading control methods. This basic scheme have been derived from the analysis of a model of virtual memory system. The control methods determine an utilization rate of the secondary memory device which is in accordance with the utilization of all the system.

Let us remark that this principle can be extended to control the utilization of the other input-output devices.

ACKNOWLEDGMENT

The author wishes to thank P.J. Courtois of MBLE Research Laboratory for the fruitful discussions about this work.

REFERENCES

[1] M. Badel, E. Gelenbe, J. Leroudier, D. Potier
"Adaptive Optimization of a Time-Sharing System's Performance",
IEEE Proceedings on Interactive Computer System, June 1975.

[2] L.A. Belady, C.J. Kuehner
"Dynamic Space-Sharing in Computer Systems"
Com. ACM, Vol.12, n°5, May 1969.

[3] A. Brandwajn, E. Gelenbe, J. Lenfant, D. Potier
"A Model of Program Behaviour in Virtual Memory"
Advanced Course on Operating Systems Principles, Pisa 20-31, Agosto 73.

[4] D.D. Chamberlain, S.H. Fuller, L.Y. Liu
"A Page Allocation Strategy for Multiprogramming Systems",
IBM Research Report RC 3848, May 1972.

[5] P.J. Courtois
"Decomposability, Instabilities, and Saturation in Multiprogramming Systems",
Comm. ACM Vol.18, n°7, July 1975, 371-377.

[6] C. Glowacki
"Contribution au Contrôle de la Charge d'un Système Informatique à Mémoire Paginée"
Ph.D.Thesis, Université Libre de Bruxelles, juin 1977.

[7] J.R. Jackson
"Jobshop-like Queueing Systems",
Management Science 10, 131-142, (1963).

[8] "Principles of Optimality for Multiprogramming"
Proc. Int. Symp. on Computer Performance Modeling Measurement and Evaluation, ACM-SIGMETRICS and IFIP WO7.3, Cambridge (Mass.), March 1976, pp.211-218.

[9] A.J. Shils
"The Load Leveler",
IBM Research Report RC 2233 (1968).

ARCHITECTURE LANGUAGE

G. Dávid
Computer and Automation Institute
Hungarian Academy of Sciences

INTRODUCTION

Architecture Language AL is an attempt to describe computing systems. The expression "computing system" is meant here as a multilevel system consisting of various levels of hardware, of firmware and of software. This requires a level-independent language to be designed. But AL is intended to use for many purposes. One of the most important goal is to describe computing systems during their development processes, and this requires documentation, verification, testing etc.

The basic ideas can be summarized as follows:

- the architecture of the system can be described component-wise

- every component can be represented as a frame, a new notion, which is a self-contained description of the component

- a frame can be used in describing another frame

- a frame can be manipulated independently from the others (test, verify, change etc.)

- a frame contains information for all of the possible manipu-

lations.

Formaly a frame is divided into three parts:

- interface

- specification

- implementation.

The interfacepart gives the formal name and the parameters of the frame. The specification says what the frame (and the component, represented by the frame in question) will do and the implementation says how the frame will be realized.

Every part can be changed during system-development, and the implementation-part maybe empty.

Consequently AL is a "three-dimensional" programming language, consisting of the languages of interfaces, of specifications and of implementations. These languages will be described here and the connections between them.

INTERFACE LANGUAGE OF AL

This is the simplest language from AL: it specifies for external use mostly the syntactical form. Every frame is written in the form of

```
frame frame name (FIPL;FOPL)
"interface";
specification
      specification-part;
implementation
      implementation-part;
endframe;
```

where FIPL is the formal input parameter list, FOPL is the formal output parameter list. The "interface" specifies the names in FIPL and FOPL, declaring by

"declarator" list of names;

where "declarator" may be bit (n), byte (n) or an arbitrary word, representing some composed type, constructed in the language of SL (1).

SPECIFICATION LANGUAGE OF AL

The purpose of this language is to describe what the frame will do. Although the logic-based language had been already designed, we feel that a high-level language is more suitable for the present-days practice. Hence we accepted the concepts of algorithmic languages, but it is important to note that the specification-part of a frame never will be executed.

Basic types are integer, real, boolean. In the declaration part the user may declare new types with

type type-name<{selector$_i$: type$_i$}>

where type$_i$ had been already declared or a basic type, and selector$_i$ specifies the selector of the subtype type$_i$. An object of this type can be declared by

type-name object-name;

and a substructure of this object can be referred as

object-name [selector$_j$],

representing a subobject of type$_j$. For homogeneous types we can use the declaration

```
type type-name<[lb:ub]: type'>
```

where lb and ub is lower and upper bound resp, each subtype is the same type'.

Every symbol, declared may have attribures, declared by

```
declarator symbol.attribute begin body end;
```

Declarator is a type-name, "attribure" is specified by the user. The body may contains every language-constructs which is allowed. For example, if A is a structure, then

```
boolean A.event1
      begin
      A. event1  : = if (A[Ci] = 0.1) then false else true;
      end
```

when A[Ci] is a substructure of A.

The symbols, declared in the interface-part, can be declared here again with apropriate type, because the specification-part states something on the content of the object, if not declared, then it is assumed as integer.

To every symbol automatically the time attribute is associated, independently from the declaration. The

```
symbol.time
```

represents it, and can be referred, manipulated, etc.

In the specification part we allow the following language constructs:

begin ... end
if ... then ... else
while ... do ... od
assignments,
declarator function function-name (formal parameters),
parameter-specification;
begin body end;

Expressions are composed from elementary functions or declared functions, logical expressions are composed from infix predicates ($<,>,=,\leq,\geq$) and logical connectives ($\wedge,\vee,\sim$) and quantifiers ($\forall,\exists$). Logical constants are true and false.

Predicates can be quantified for time-interval also by

$$[\text{time-expression}_1, \text{time-expression}_2]$$

where time-expression is an arbitrary expression, for example

$$(C \geq 50)[-20,(A.time)+2]$$

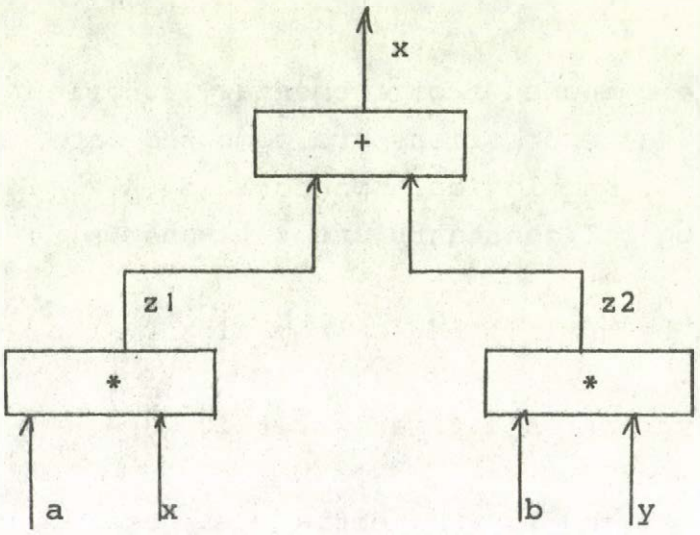
The time-constant "0" is the time of the activating of the frame, and every time compared with this time. The predicate in the example is evaluated as true if in the interval the predicate is true, else false.

IMPLEMENTATION LANGUAGE OF AL

The implementation language is a data-flow language [2], modified for systems-development purposes. In data-flow languages there is given a set of functions, represented by boxes in the figure and input and output lines. If the functions for example are * and +, then

```
*(a,x;z1)
+(z1,z2;w)
*(b,y;z2)
```

is a program of a*x+b*y, because the only symbol which is not used as input is the w, the symbols, used as input only are a, x, b and y, hence we have



In data-flow languages one function is executed if and only if all of its input has been arrived. It should be also noted that the program above is only an unordered set of statement - the ordering is given by the topology and the self control mechanizm.

For our purposes, frames play the role of the functions, but the control is a maniputable object like data items. The reason to do so in that during system-development the programmer can not design the control before the implementation, but only after a carefull testing. Hence we had developed a control--language, [3,4] and the control can be described (based on Petri-nets).

But also the implementation-part is consisting of different parts:

declarations;
labelled modules;
control-algorithms;
control-action;

In the declaration-part the internal, local variables can be declared and those frames which will be used in the implementation. This can be done by declaration

frame frame-name (actual parameters)

or by

frame-name actual-name (actual parameters)

The first declares only one copy of the frame, with the original frame-name, the second allows us to use many copies of the frame with different actual parameters and with different names as "actual-name".

Labelled modules are those parts of the implementation which controlled by the control-action. Control-action is an activation of that control which had been computed by control algorithms. One can interpret of the notion of the module as a (maybe locally controlled) program, which will be executed as it would be a data-flow computer. But from the point of view of the whole frame, it is a sequentially executed program-part.

Formally a module is written as

label (parameters):
logical-expression → module-body

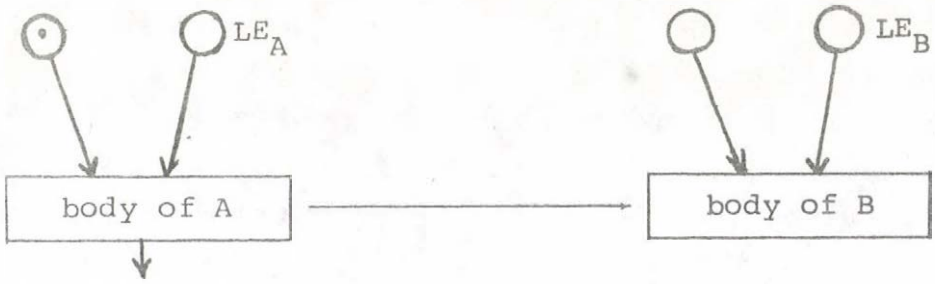
where

label - arbitrary (but unique) symbol, local to the frame

parameters - integer variables (but maybe empty)
logical expression - constructed from predicates (with time-quantification if necessary) as in the specification language.

Their role is the following: A module can be referenced by its name and actual parameters. The parameters if used in module--body will be substituted by the actual ones. The system routes the control to this module, if the control-action does and the logical expression is true. Logical expression may couse wait, if when the module was activated then it had been evaluated for false but by the time its value has been changed to true.

The module-body is a list of actual frames. The actual frames are checked by types and by attributes. Module-body describes topologicaly the flow of information and the transformations executed by the frames. It is possible between the two modules the information flows, but it does not mean the execution of the other. Using the graphic representation of Petri-nets, then if we have two modules A and B with logical expression LE_A and LE_B



and there is an information-interchange between the bodies, for example A sends something to B, then it will be used only if the own control of B will activate it.

The description of control-structures is published elsewhere. AL is in the design phase, during implementations some parts will be changed.

REFERENCES

1 G. Dávid, S. Keresztély, I. Losonczi, A. Sárközy: Logic-
-based Description of Microcomputers
MTA SzTAKI Tanulmányok 91., pp. 75-92, 1978

-: Microprogram Synthesis, ibid, pp. 187-206.

2 J.B. Dennis: First version of a data flow procedure language
Lecture notes in computer science, 19,
G. Goos, J. Hartmanis (eds) Springer-Verlag, 1974,
pp. 241-271.

3 G. Dávid: On the Basic Component of a Module Language
MTA SzTAKI Tanulmányok

4 G. Dávid: Description of Dynamic Control Structures
Proc. of ALGORITHMS'79, Strbeske Pleso, Czecho-
slowakia, pp. 241-252.

A PROTECTION MECHANISM FOR MODULAR SYSTEMS

Gábor HROTKÓ, Gábor SIMOR
Institute for Coordination of Computer Technics
Hungary

Abstract.

A capability-based protection mechanism is proposed for systems consisting of modules with abstract data type definition. A run-time access control is assumed, which is especially reasonable in real time systems with high reliability.

Access rights are specified in the procedure specification part and the component object definition part of the modules for different type definitons. Entering a new module via procedure invocation causes either switching or amplifying of the acces right domains depending on the nature of the type: whether it is only an object or an object and a subject at the same time. The control for acces right correctness is concentrated in two main system functions: creation of new instances for a given type and invocation of procedures.

The synchronization correctness is not considered in the paper, however interpretation of the conditions - defined by path expressions - is proposed to be implemented by checking and modifying of the acces right domains.

1. Introduction

The concept of the capability-based protection model /a brief definition of the basic n tions see below in 2./ was founded by B.W.Lampson /[5]/ and it has been used in most of the works dealing with the protection aspects of operating systems, programming languages and computer architectures /eg. [1] - [10] /.

A.K. Jones in several papers / [1] , [2], [10] / has pointed out the relevance of this approach of access control to the object type oriented languages, like SIMULA, CLU, ALPHARD.

In [2] object name binding rules providing acces control correctness have been established giving a general approach for language extensions.

A similar, but in certain aspects different approach was published by Ancilotti and others [3] . They presented features of a Concurrent Pascal - like language, providing access control even in the case of dynamic resource allocation. In both of the above-mentioned works compile-time access control checking was supposed in order to avoid efficiency losses in execution. However in many aspects a run-time acces control may be reasonable:

- effects of any software or hardware execution malfunction /i.e. not only implicitly erroneous access altempts/ may be localized in a small domain exercising access control: effect of the "principle of least privilege"([3]) is thus promoted;

- access rights on /and for/ dynamically created objects are also controllable;

- dynamic passing of access rights - dependent on execution-time events -are also controllable: dynamic allocation of resources doesn't force the programmer to describe distinct "manager" modules, as in [3] ;

- contemporary multiprocessor computer structures offer possibilities for the use of special access control modules, as proposed in [4] .

Several papers have proposed solutions for run time access control([4], [7], [8], [9]), however none of them has shown, how the control mechanism is embedded into a modular language /or perhaps nonlinguistic modularly structured/ system, like the language features that were specified in [3]. Also several papers have pointed out the relevance of the capability-based access control to the "path expression"-oriented synchronization approach ([1], [3], [7], [10]), however none of them have shown a mechanism for both protection and synchronization.

The aim of this paper is to define an access control mechanism, capable of

- embedding it into an ALPHARD-like object-type oriented modular language, containing the relevant access control and synchronization features, offered in the previous works ([1], [2], [3]);

- controlled dynamic creation of object instances and providing the other adventages of run-time control;

- implementation of unique tools for controlling protection and synchronization conditions.

2. Components of the Protection Model

The proposed protection mechanism is associated with the following basic notions, which were also used in similar sense in [1] ÷ [6]:

Object: an entity to which access must be controlled;

Subject: an entity whose access to objects must be controlled;

Capability: an access right a subject can excrcise in order to invoke an operation on an object

Domain: a set of capabilities that one subject has to the objects of the system

Using these notions Lampson has proposed [5] the protection models to be represented in the form of a protection matrix, rows of which are associated with domains /subjects/ and columns with objects /Fig.1/

Domain switching: an action which causes a substitution of a valid protection domain /moving from one row of the protection matrix to another/

2.1. Notions in a Type Oriented Protection Model

Type: a class of objects with the same behavior: the same operations with the same effect /and only those operations/ are allowed for accessing the objects; the same access and synchronisation constraints are defined for them,

Type module: a program fragment which defines a type;

Instance: an actual object /created using a type module as a template/;

Qualified type: a type with the subset of its associated operations. The operations contained by the qualified type are the only ones albwed on the object described by the qualified type. /A qualified type defines the content of the corresponding crosspoint in the protection matrix/.

Binding: assignment of a name with qualified type to an object: binding also means changing acces rights /capabilities/ for the given object /eg. in the case of domain switching/

Binding rule: normally a binding is legal if
- types in the old and the new qualified types are similar;

- the set of operations in the new qualified type is the same as in the old qualified type or it is the subset of the second.

The binding rule may be avoided in two ways:

- domain switching;
- amplification /revocation/

Amplification: at the invocation of an operation /Revocation/ on an object the current domain may be amplified /revoked/ by capabilities defined inside the type module which has served as a template for the creation of the object.

3. Defining Type Modules

In our model a type module contains the following protection-relevant information[x] (taking into consideration proposals in [1]÷[3], [11]):

- In the specification part:
  - name of the type [xx]
  - names of the operations, specified for the given type of objects. For every operation:
    - list of the parameters. For every parameter:
      - the qualified type supposed before the invocation of the operation /the new qualified type for binding/;
      - the qualified type supposed after the invocation of the operation.
  - the kind of the type module regarding domain modifications: whether the domain is switched or amplified at the operation invocations

x We dont consider here the correctness proving aspects of the type modules (see ALPHARD forms in [11]): a simplified description is presented.

xx We apply another simplification: parameterless types are supposed. Handling of type parameters requires additions in the creation mechanism described below in 5.

- path expressions, defining synchronization among the operations specified here /see below 5./

- In the representation part:
  - list of the component objects, needed for implementation of the specified operations and also created at the creation of an instance of the qiven type[x]. For every component object:
    - the qualified type, defining capabilities needed for the execution of the operations specified in the type module[xx]
  - the initial state of the object
- In the implementation part
  - implementation of the operations, specified in the specification part, induding also invocations of operations on objects, defined in the representation part. The parameters of the infokted operation is either on object declared in the representation part or is the parameter of the implemented operation itself.

xx For simplicity, we consider the same access rights for the whole moduleAnother possibility wald be assignment of different qualified types for different operations in order to promote the "principle of least privilege".

x We make no difference here between the unique and the common component objects.

A simplified syntactical definition of a type module may be presented as follows:

type module ⟨type name⟩

specification

operations

⟨1-st operation name⟩

	type	required rights	given rights	revaked rights
input / output	⟨qualified type lists for required, given, revoked rights /possibly empty/ connected to the parameters⟩			

⋮

⟨n-th operation name⟩

input / output ⟨parameters with qualified type lists⟩

domain ⟨switched or amplified⟩

path restrictions ⟨path expressions⟩

representation

components ⟨list of object names, each with its qualified type⟩

initially ⟨initial conditions⟩

implementation

⟨1-st operation name⟩
:
invoke (⟨component object name⟩: ⟨type name⟩ ,
⟨operation name⟩,⟨list of parameters ,
each with its type⟩)
:
⟨n-th operation name⟩

end

Defining type modules in this way imply the following consequences:

a./ a program may be conceived as
- a set of type modules with an invocation hierarchy among them;

- an initial creation statement for an instance /of the outermost type module/;

- an invocation of an operation on the previous initially created instance.

The specification and the representation part of a type module provides all information, needed for creation of an instance. The instance creation mechanism by its recursive execution provides the creation of all of the initially needed instances for the program after the initial call for it.

b./ a type module provides a definiton for both object and subject types. A type module defines a subject with its own stand-alone domain if a switched domain is declared in its specification part.

In case an amplified domain is declared the capabilities of the calling subject's domain are also inherited.

c./ binding rule providing access control correctness has generally two main applications: variable declarations, procedure invocations([1],[2]).

In the proposed type module environment only the latter has meaning for the protection mechanism: the correct use of variables declared in the representation part by the implementation part may be trivially checked at compile time inside the same module.

For the operation invocations a new qualified type is defined as required rights in the specification of the invoked operation;
an old qualified type is defined in the calling module either in the representation part as a variable declaration or in the specification part as required rights.

d./ Kinds of domain amplifications and revocations and domain switching in the type module environment may be summarized as follows in the table 1.

Qualified type declaration	Domain amplification: Amplification	Domain amplification: Revocation	Domain switching
In the representation part	At the beginning of any operation invocation	At the end of any operation invocation	The new domain contains the corresponding capabilities and is valid during the execution of any invoked operation
As required rights	no /only checking/	no	The new domain contains the corresponding capabilities if the given operation is invoked and if the old domain has also contained them
As given rights	At the end of execution of the given operation	no	The old domain is amplified after the given operation has been executed /and the domain reswitched/
As revoked rights	no	At the end of execution of the given operation	The corresponding capabilities are revoked in the old domain after the given operation has been executed

Table 1.

The choice of domain amplification or domain switching at the type module definition may be decided taking into consideration the following circumstances:

- domain switching is more suitable in the viewpoint of the "principle of least privilege";

- domain switching demands a domain saving mechanism;

- domain apmlification provides less reliability but more flexibility: certain objects accessable for a higher-level module may be legally accessed also by certain lower-level modules /with appropriate relation in the invocation hierarchy/ without implicitly passing them as parameters

4. The operation of the Protection Mechanism

The protection mechanism of a type module environment itself may be conceived as part of the functions of a special built-in type module. This module defines the type: intstance. We can suppose, that the "instance" type module has generally only one instance, which statically exists and hasn't been initially created.

The instance type module defines various operations, four of which are connected also with access control functions: create, invoke, return, destroy. Here we present the specification of /the access control relevant functions of/ the create and invoke operations. In the specification a simple predicate logic based notation is used, with the following functions (see *Fig.2*):

module (I:T) - the instance I of type T has been created

{comp.objects (I:T)} - the set of component objects of the instance I defined in the representation part of the type module T

capability(I : T, I1:T1,OP) - for the domain of the instance I /subject/ with type T an acces right is granted to invoke operation OP on the object I1 with type T1

{qualified type(I)} - the set of access rights defined by the qualified type of the instance I

operation (T,OP,PL)- in the type module T the operation OP with /typed/ parameters PL has been defined

current-domain (I:T)- the instance I with type T is the currently valid domain

{rights-required (T,OP)} - set of access rights defined in type module T as required rights for invocation of the operation OP

OP (I:T, PL) - the instance name I with parameters PL has been passed immediately to the module T in order to execute operation OP

The operations return, destroy may be specified in a similar way.

5. Extensions for Synchronization Control

"Path expressions" are the most natural means of formulating synchronization constraints in a type module environment. Interpretation of the path expressions may be carried out in the four operations-ment oned in the previous section - in a similar way, as the access constraints have been: at the invocation of certain operations some others should be forbidden /capabilities revoked/, after the execution of the operation they may be again allowed /capabilities restored/ and new operations also allowed /capabilities added/.

The only differences are in the actions to be performed in the case of any attempt to invoke an operation beyond the valid capabilities of the domain. In the case of violating access rights the occurance of a "hard" exception is assumed, synchronization constraints may cause queueing actions.

6. Conclusions

The "amplification-oriented" approach([1], [2]) and the "domain switching-oriented" approach([3]) of access control in a modular, abstract data type definition environment may be succesfully combined. The binding rule, amplification, revocation and domain switching together are included in the protection mechanism, the access constraints are specified merely at the operation and the component object specification in the type definitions.

The access control may be reasonably extended also for run-time functions and concentrated in a few operations on type instances. The same mechanism may have a natural extension for handling of "path expressions".

7. References

[1] A.K.Jones: The Narrowing Gap Between Language Systems and Operating Systems IFIP'77 p. 869-873

[2] A.K.Jones, B.Liskov: A Language Extension for Expressing Constraints on Data Access. CACM May 1978 p. 353-367

[3] P.Ancilotti, M.Boari, N.Lijtmaer: Language Features for Access Control
Istituto di Automatica Universita' di Bologna
Report N.37 - April 1978.

[4] J.Gorski: A Modular Representation of the Access Control System
SIGOPS July 1978 p. 61-77

[5] B.W.Lampson: Protection. Proc. Fifth Annual Princeton Conf. on Information Sciences and Systems, 1971.

[6] A.V.Jones: Protection in Operating Systems Lecture Notes in Computer Science Vol.60.1978.p 228-251.

[7] L.Boi, P.Michel: An Approach to a Fault-Tolerant System Architecture. The 5th Annal Symposium on Computer Architecture. Conf.Proc.1978.p. 123-130.

[8] H.J.Saal., I.Gat: A Hardware Architecture for Controlling Information Flow. The 5th Annual Symposium on Computer Architecture Conf. Proc. 1978. p. 73-77.

[9] G.J.Battarel, R.J. Chevance: Design of a High level Language Machine. Computer Architecture News 1978.

[10] A.K.Jones: The Object Model: A Conceptual Tool. Lecture Notes in Computer Science Vol.60. 1978. p. 7-16

[11] W.A.Wulf, R.L.London, M.Shaw: Abstraction and Verification in ALPHARD: Introduction to language and methodology. Techn.Report Carnagie--Mellon University, Pittsburgh 1976.

Fig 1. A protection matrix

	O_1	O_2	. . .	O_i	. . .	O_n
D_1	β	γ				ξ, ψ
...						
D_j	α	δ, ε		μ		
...						
D_m	α, β					ψ

objects

access rights (operations)

Domains

Type module instance
specification

operations	object name	type	required rights	given rights	revoked rights
create					
input	<instance name>	instance	create	destroy invoke	—
	<type name>	type modul	read component objects	—	—

pre: modul (<instance name>: <type name>) = ↓

post: [modul (<i.n.>$_i$: <t.n.>$_i$) = ↑ | if <i.n.>$_i$: <t.n.>$_i$ ∈ {comp.objects (<instance name>: <type name>)}] ∧

[capability (<instance name>: <type name>, <i.n.>$_i$: <t.n.>$_i$, <operation>$_j$) = ↑ | if
<i.n.>$_i$: <t.n.>$_i$ ∈ {comp. objects (<instance name>: <type name>)} ∧
<t.n.>$_i$ – <operation>$_j$ ∈ {qualified type (<i.n.>$_i$)}]

capability (<instance name>: <type name>, instance, revoke>) = ↑

modul (<instance name>: <type name>) = ↑

Fig. 2. Specification of the „instance" type

invoke

input:	<instance name>	instance	invoke	–	–
	<type name>	type modul	read operation read domain read component objects	–	–
	<operation name>	operation	read rights required	–	–
	<parameters>	typed objed list		–	–

pre operation (<type name>, <operation name>, <parameters>) = ↑ ∧
[capability (<in>$_c$: <tn>$_c$, <in>$_i$: <tn>$_i$, <op>$_j$) = ↑ | if
current domain (<in>$_c$: <tn>$_c$) = ↑ ∧ <in>$_i$: <tn>$_i$ ∈ {<parameters>}
<in>$_i$: <tn>$_i$ – <op>$_j$ ∈ {rights required (<type name>, <operation name>)}]

post [current domain (<instance name> : <type name>) = ↑ | if
switched domain (<type name>) = ↑] ∧
[capability (<instance name> : <type name>, <in>$_i$: <tn>$_i$, <op>$_j$) = ↑ | if
switched domain (<type name>) = ↑ ∧ <in>$_i$: <tn>$_i$ ∈ {<parameters>} ∧
<in>$_i$: <tn>$_i$ – <op>$_j$ ∈ {rights required (<type name>, <operation name>)}]

Fig. 2. (continuation).

[capability (<in>$_c$:<tn>$_c$, <in>$_i$:<tn>$_i$, <op>$_j$) = ↑ | if
amplified domain (<type name>)↑ ∧ current domain (<in>$_c$:<tn>$_c$)=↑ ∧
capability (<instance name>:<type name>, <in>$_i$:<tn>$_i$, <op>$_j$) = ↑]
<operation name> (<instance name> : <type name>, <parameters>) = ↑

Fig. 2. (continuation).

SIMULATION STUDIES AT NPL OF FLOW CONTROL AND ROUTING IN PACKET-SWITCHED NETWORKS

Wyn L. Price, United Kingdom

Design of a data communication network to a given specification, involving a number of nodes and links, is well-known to be a non-trivial exercise. Not only may a particular proposed target performance specification be unattainable or attainable only at inordinate cost, but the sheer intellectual difficulty of predicting the performance of what appears to be a feasible design solution can be formidable. The human designer needs to be able to call upon appropriate tools or aids to help him in the task both at design and evaluation stages. Such tools have been highly developed during the relatively short history of data communication networks. They fall into two main categories, those employing queueing theory and network flow theory and those involving computer simulation. It is beyond the scope of this short note to discuss the merits and drawbacks of theoretical methods; an excellent guide to these may be found in Kleinrock (1).

If it is necessary to model network performance taking into account details of operating systems and protocols, especially those controlling some limited resource, then

simulation may be the only tool left to the experimenter. In a simulation model the experimenter has the freedom to express the subject modelled, in this case, communication networks, to whatever degree of detail is necessary.

Studies of network routing by simulation

Amongst the earliest subjects to be studied by simulation methods was the evaluation of routing algorithms for data communication networks. The problem is a difficult one, not easily amenable to treatment by analytic methods. The practical design problem pivots around the need to make local routing decisions based on incomplete up-to-date knowledge of the state of distant parts of the network. The "out-of-dateness" of the information acts a phase lag in what is effectively a complex control loop, leading to possible instability.

The merits of rival routing doctrines have been rehearsed elsewhere (2) and a full discussion would be inappropriate in the present paper. Suffice it to say that the choice is broadly between fixed and adaptive policies, and that adaptive policies can be further subdivided according to the manner in which they monitor network behaviour and the manner in which routing decisions are implemented. The monitoring process may be local, distributed or centralised, as may the decision process.

An early simulation study of shortest path and other routing doctrines was carried out by Plessey (3), under contract to the UK National Physical Laboratory (NPL). Attempts by simulation to show that adaptive routing of one kind or another was superior to fixed routing had earlier produced negative results. Therefore an experiment was devised in which each node, when taking a routing decision, was provided, as though by magic, with a complete overview of the whole network. The state of all nodes and links, congested or otherwise, could therefore be taken into account in computing for each packet a route of minimum delay. The unexpected result was to show no improvement for magic routing over the performance obtained with fixed shortest path routing, performance being expressed in terms of maximum overall network carrying capacity and mean delay per packet in transit through the network. It was concluded that routing decisions which were optimal when they were taken became suboptimal, because of changed circumstances, before the routed packets reached their destination. Another cause could be a tendency for several nodes simultaneously to route traffic towards what was temporarily an underloaded section of the network, causing this section to receive too much traffic and to become overloaded in its turn. This is evidently an unstable condition.

Fixed routing should not be dismissed out of hand as a routing doctrine. Other experiments at NPL (4) have shown that, given a constant traffic matrix applied to the network, a fixed routing matrix can be devised that optimises network performance; this fixed routing matrix is developed by an iterative process from the shortest path matrix. It is when the routing algorithm needs to account for a changing load balance or for network component outages that fixed routing becomes a less attractive proposition.

On the other hand, it has long been believed by various workers that adaptive routing, properly applied, can enhance ultimate network carrying capacity or reduce transit delays at heavy loads. The ability of certain forms of adaptive routing to split loads over more than one route was expected to give this benefit; an early load splitting algorithm was the "bifurcation" algorithm discussed by Fultz (5). A bifurcation routing algorithm was simulated at NPL (6), the proportion of traffic splitting in each node being governed by route topology and by the length of packet queue waiting to use each route. The ultimate performance of a network thus organised was found to be inferior to that of a network with fixed routing matrix "tailored" to the traffic

matrix. This result was explained on the grounds that under heavy general load there is little or no spare capacity which can be used to provide an alternative route.

On the other hand, when the same load-splitting algorithm was simulated on the same network under conditions of a moderately heavy general load with an additional single heavy stream of traffic from a particular source to a particular destination, the ability to share traffic between routes produced a striking improvement in performance (7). We conclude, therefore, that load -splitting can be very effective for dealing with isolated heavy traffic demands, but contributes little, if anything, when the heavy demand is general.

Simulation studies of flow control and congestion avoidance

Flow control operates at various levels in a packet -switched network. It is concerned to ensure that, in any flow of traffic between two points, the input flow shall not, averaged over some appropriate period, exceed the output flow. Should the input flow substantially exceed the output flow for a significant period of time, then a build-up of traffic is inevitable within the communication medium; if the latter has a limited capacity, then congestion may occur, followed by a possible complete breakdown of the system.

Flow control may be, and usually is, applied between adjacent network nodes connected by single links. Flow control may operate between source and destination network nodes. It may operate between source and destination host computers or between processes contained within these computers. In some networks flow control is installed at all these levels, in others it may be found only at one or two of the levels listed.

The simulation work at NPL has included studies of flow control in a complete sub-network context. The major results concern networks operating in datagram mode, though the method could easily have been extended to include end-to-end protocols, including virtual calls.

Details of the early work at NPL may be found in (8) and it is sufficient to remark here that the systems simulated were prone to congestive failure when excessive loads were applied to the source interfaces of the network; this was due in part to the nature of the adaptive routing algorithm in use and in part to the lack of an effective flow control on input. The node pool of output buffers was divided equally between all the output queues (with also a fixed allocation per input queue). The adaptive routing algorithm (rather like Baran's (9) "hot potato" routing algorithm) operated by seeking the best output

queue for each packet from a list, according to packet destination, giving the ranked order of all the node output queues. Thus a slot would be found on some output queue, however unsuitable, even if that slot represented the last available output buffer in the node. Application of a general heavy load to such a network rapidly filled up all the node buffers. Only when all the buffers were full was input traffic cut off. The result under heavy applied load was rapid saturation of the network, with packets unable to progress towards their destinations.

Work by Plessey (10) for NPL showed that optimum performance of the network could be obtained by placing an artificially low limit on the number of packets allowed into the network at one time. Davies (11) proposed a system whereby such a low limit could be maintained in practice - this was the system known as "isarithmic" flow control, which was studied extensively by simulation methods at NPL (4,12). In an eighteen node network with over 400 buffer store locations throughout the network, best performance in terms of throughput and delay was obtained when not more than fifty packets were in transit at one time. Application of isarithmic control to the network already shown to be prone to congestive failure removed all risk of such failure, however great the level of external load applied. The isarithmic scheme depends

on the provision of permits which packets seeking admission need to acquire; permits are given up when packets are delivered to their destinations.

Other NPL work (13) showed that if the control on packet admission was made positive enough by other means, then the isarithmic scheme was not needed. Such a positive control was afforded by a system based on node buffer occupancy. If an incoming packet could not be found a buffer, then it was made to wait for admission; all incoming packets were blocked until this condition was relieved. Packets already travelling within the net (arriving in a node from neighbouring nodes) were given a measure of priority over new incoming packets by being allowed access to reserved buffers which was denied to new packets.

Both these schemes, isarithmic flow control and buffer occupancy control, are distributed schemes which are capable of maintaining a network in a non-congested state. A close parallel exists between this NPL work and other work at GMD (14).

It is fundamental to good network flow control that the first priority of a network shall be to seek to deliver traffic already accepted for transmission and that acceptance of new traffic must take second place.

Simulation of a hierarchical network

One method of coping with the organisational problems of very large networks is to arrange that the topology and control system shall be hierarchical in nature. In a hierarchical network nodes would be arranged in groups, each group served by a super node; super nodes would be connected together (not necessarily fully connected) by a number of fast transmission links; super nodes would be grouped into hyper groups each served by a hyper node; hyper nodes might be fully connected by even faster links. The depth of the hierarchy should depend on the size of the network and on the degree of grouping (number of nodes in one group) considered desirable.

A study of flow control and routing in such a hierarchical network has been carried out by Logica (15,16) under contract to NPL. This study has shown that the control protocols previously found adequate in a homogeneous network need strengthening in the hierarchical context. For example, flow control by source node buffer occupancy was found inadequate in the hierarchical network. This was because congestion could occur in a higher level of the network and was only weakly fed back to the lowest level nodes responsible for controlling traffic acceptance from the network subscribers. A satisfactory solution was found in the re-introduction of isarithmic control in

the higher level of the network. An attempt to imitate in the hierarchical store-and-forward network the routing doctrine used in the hierarchical circuit-switched telephone network failed to show satisfactory performance. The most satisfactory routing algorithm found was one based on local queue length information, such as has already been discussed in this paper.

It seems beyond doubt that hierarchical organisation will be necessary in the very large networks of the future. Network control problems are likely to be rather different from those encountered thus far in non-hierarchical contexts. Simulation techniques will of necessity be required for the study of these very complex issues, but the techniques themselves may need adaptation to take account of the very large scale of the networks involved.

References

1 Kleinrock, L. Queueing Systems. Vol. 2: Computer Applications. Wiley, New York, 1976.

2 McQuillan, J.M. Adaptive routing algorithms for distributed computer networks. Bolt, Beranek & Newman, Report 2831, May 1974.

3 Adams, R.A.. Jolly, J.H. & Smith, J.R.W. Simulation study of a data communication network, part 3: comparison of adaptive and fixed routing. Plessey Telecommunications Research Ltd., Report 97/73/03/TR, February 1973.

4 Price, W.L. Simulation studies of an isarithmically controlled store and forward data communication network. Proc. IFIP Congress 74, Stockholm, August 1974, 151-154.

5 Fultz, G.L. Adaptive routing techniques for message switching computer communication networks. University of California, Los Angeles, Report UCLA-ENG-7252, July 1972.

6 Price, W.L. A study of bifurcated routing in a data network and the effect of isarithmic control in this context. NPL Report COM72, March 1974.

7 Price, W.L. Adaptive routing in store-and-forward networks and the importance of load-splitting. Proc. IFIP Congress 77, Toronto, August 1977, 309-313.

8 Healey, R. Computer network simulation study. NPL Report COM64, January 1973 (originally issued in earlier form, October 1970).

9 Baran, P. On distributed communication networks. Trans. IEEE, CS-12, (1), March 1964, 1-9.

10 Jolly J.H. & Adams, R.A. Simulation study of a data communication network, part 4: second system description and results. Plessey Telecommunications Research Ltd., Report 17/73/06/TR, February 1973.

11 Davies, D.W. The control of congestion in packet-switching networks. Trans. IEEE, COM-20, (3), June 1972, 546-550.

12 Price, W.L. Data network simulation, experiments at the National Physical Laboratory 1968-76. Computer Networks, 1, (4), May 1977, 199-210.

13 Price, W.L. Simulation studies of data communication networks operating in datagram mode. Computer Journal, 21, (3), August 1978, 219-223.

14 Price, W.L. & Haenle, J.O. Some comments on simulated datagram store-and-forward networks. Computer Networks, 2, (1), February 1978, 70-73.

15 Kerr, I.H., Gomberg, G.R.A., Price, W.L. & Solomonides, C.M. A simulation study of routing and flow control problems in a hierarchically connected packet switching network. Proc. ICCC, Toronto, August 1976, 495-501.

16 Gomberg, G.R.A., Kerr, I.H., Richards, J., Price, W.L. & Solomonides, C.M. A design study of a hierarchically connected packet-switching network using simulation techniques. Computer Networks, 3, (2), April 1979, 114-135.

Dr. Wyn L. Price is a member of the staff of the Division of Numerical Analysis and Computer Science, National Physical Laboratory, Teddington, Middlesex, UK.